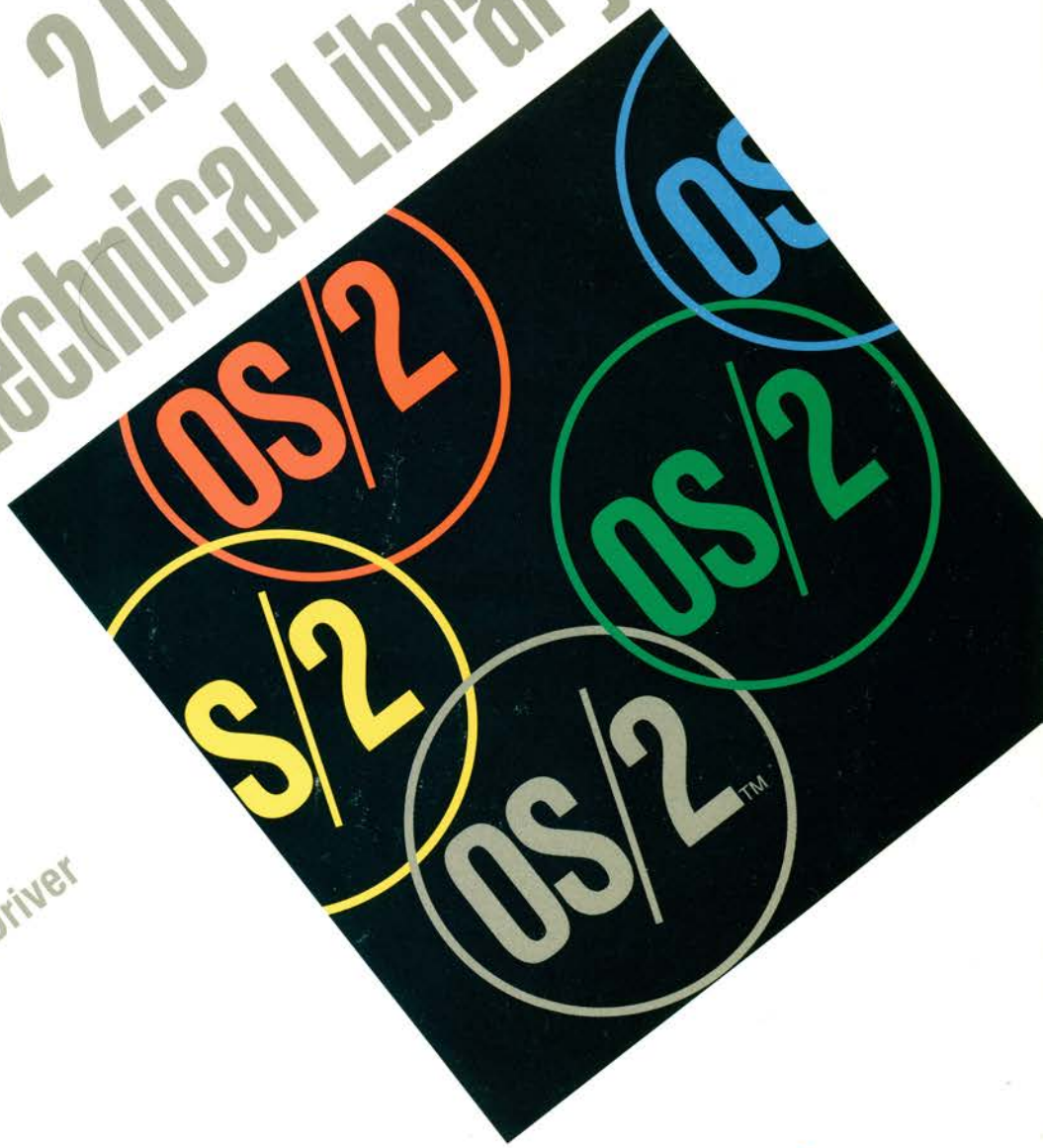
OS/2 2.0 Technical Library



Virtual Device Driver Reference

Version 2.00

IBM Programming Family

OS/2 2.0 Technical Library

Virtual Device Driver Reference

Version 2.00

Programming Family

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page ix.

First Edition (March 1992)

Contents

Part 3. Reference Material

Notices

Trademarks

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States and/or other countries:

IBM
PS/2
OS/2
Presentation Manager

The following terms, denoted by a double asterisk (**) in this publication, are trademarks of other companies as follows:

Intel	Intel Corporation
Lotus	Lotus Corporation
Microsoft	Microsoft Corporation

Double-Byte Character Set (DBCS)

Throughout this publication, there are references to specific values for character strings. These values are for the Single-Byte Character Set (SBCS). When using the Double-Byte Character Set, note that one DBCS character equals two SBCS characters.

About This Book

The *OS/2 2.0 Virtual Device Driver Reference* defines what a virtual device driver is, how it operates, and when to use one. In addition, a description of the types of virtual device drivers, their interfaces, and the available kernel services is provided. System and application programmers can use the information found in this book to write their own virtual device drivers and subsystems.

Knowledge of at least one programming language that is used for writing OS/2 applications is necessary, and the programmer must be familiar with the workings of the OS/2 operating system. The programming concepts that should be understood before developing applications to run on OS/2 2.0 are found in the *OS/2 2.0 Application Design Guide*.

"OS/2," as used in this book, refers to Version 2.00 of the OS/2 operating system unless stated otherwise.

This book consists of three parts: An overview of virtual device drivers (Part 1); a breakdown of the types of OS/2 2.0 virtual device drivers (Part 2); and a detailed reference section (Part 3). A more detailed description of the contents follows.

Part 1. Overview

Chapter 1. Introduction
This chapter contains a general description of the three types of OS/2 device drivers, and an overview of the virtual device driver mechanism and kernel services.

Chapter 2. Virtual Device Driver Architecture and Operations
This chapter contains a description of virtual device driver architecture, operations, and inter-device driver communications.

Part 2. OS/2 2.0 Virtual Device Drivers

Chapter 3. Base Virtual Device Drivers
This chapter contains a description of each base virtual device driver sent with OS/2 2.0.

Chapter 4. Installable Virtual Device Drivers
This chapter contains a description of each installable virtual device driver sent with OS/2 2.0.

Part 3. Reference Material

Chapter 5. Virtual DevHlp Services
This chapter contains descriptions of the kernel functions available to virtual device drivers.

A glossary and an index are included at the back of this book.

Part 1. Overview

Chapter 1. Introduction

Virtual device drivers are used by the OS/2* 2.0 operating system to act as virtual devices for DOS applications executing in a DOS Session. These device drivers provide *virtual hardware* support for DOS and DOS applications.

The virtual device drivers used for device virtualization are shipped with OS/2 2.0 for most of the standard devices, including:

- Asynchronous Communication (RS-232C)
- DMA (Direct Memory Access)
- Fixed Disk and Diskette
- Keyboard
- Mouse
- Parallel Port Printer
- PIC (Programmable Interrupt Controller)
- Timer
- Video.

This book provides an overview of the virtual device driver mechanism and the available kernel services.

Types of OS/2 Device Drivers

Three types of device drivers are used in OS/2 2.0:

- Virtual device drivers
- Physical device drivers
- Presentation drivers.

Virtual Device Drivers

The virtual device driver is an installable module responsible for virtualizing a particular piece of hardware and associated ROM BIOS in the manner expected by a DOS application. This device driver achieves virtualization by emulating I/O port and device memory operations. Virtual device drivers are 32-bit device drivers, which operate at Ring 0. To achieve a certain level of hardware independence, a virtual device driver usually communicates with a physical device driver in order to interact with hardware.

Physical Device Drivers

Standard I/O devices are supported by base physical device drivers that are part of the OS/2 operating system. Additional, or replacement, physical device drivers can be loaded to extend the control provided by the base device drivers or to support nonstandard I/O devices. Typical examples of these loadable device drivers are ANSI.SYS and ANSI.DLL, which are loaded by DEVICE= statements in CONFIG.SYS and provide additional functions on the interfaces to the screen and keyboard. Physical device drivers are initialized at Ring 3 and operate at Ring 0, the most privileged level of the operating system.

Further information on physical device drivers, physical device driver interfaces (including detailed descriptions of the calling conventions), and the system services available to these drivers is found in the *OS/2 2.0 Physical Device Driver Reference*.

* Trademark of the IBM Corporation

Presentation Drivers

The Presentation Manager* I/O interface for output devices is a high-level interface. This interface is similar to the API (Application Programming Interface) call interface, which uses the program stack to communicate with, or pass parameters to, the presentation drivers. These drivers are special purpose I/O routines operating with I/O privilege at privilege level 2 (Ring 2) or privilege level 3 (Ring 3). Their main function is to process function calls made by the Presentation Manager interface on behalf of Presentation Manager applications. Hardcopy presentation drivers communicate with OS/2 device drivers through the file system emulation functions. Display presentation drivers interface directly with the hardware.

Presentation drivers are dynamic link library modules that are supplied as files and identified by the extension, DRV. When the Presentation Manager is initialized, the display presentation driver is loaded and enabled automatically. Other presentation drivers (for example, the hardcopy presentation drivers) are loaded and enabled when an application calls the DevOpenDC function to open the device.

Presentation drivers service requests only from applications running in Presentation Manager Sessions in the OS/2 mode. Output data and requests for information are passed to the presentation driver as function calls to the presentation driver interface. The definition of the call interface is given in terms of the codes and data passed to the presentation driver interface through the program stack.

Include (.INC) files are shipped with the *Developers Toolkit for OS/2 2.0 (Toolkit)* to provide support for building presentation drivers that are written in assembler language. These files contain function prototypes, defined values, and data structures used by the various functions.

Further information on presentation drivers, presentation driver interfaces (including detailed descriptions of the calling conventions), and the system services available to these drivers is found in the *OS/2 2.0 Presentation Driver Reference*.

When to Use a Virtual Device Driver

Device virtualization provided by the OS/2 operating system can be extended by creating a virtual device driver (VDD) and corresponding physical device driver (PDD). A user-supplied virtual device driver virtualizes the hardware interfaces of an option adapter or device, usually to migrate an existing DOS application into the OS/2 DOS environment.

Notice that a virtual device driver is needed only in a limited number of cases. If there is no requirement for multiple sessions to share access to a device, then a requirement for a virtual device driver is unlikely. OS/2 2.0 maps interrupts, I/O ports, and I/O memory directly to any DOS device driver or DOS application which attempts to access these resources. As long as a resource is not already claimed by another session or by the OS/2 kernel, it is directly accessible to the DOS program.

Where access sharing is required, it is generally not necessary to create a virtual device driver if the I/O is handle based. DOS INT 21H I/O requests are routed by the OS/2 file system to the protect-mode physical device driver. If all I/O is handle based, it is necessary only to create the physical device driver.

For block devices, it is recommended that OS/2 2.0 support be provided only by a physical device driver. Both DOS and OS/2 Sessions will then access the device through file system interfaces defined by DOS and the OS/2 operating system.

In the absence of a sharing requirement, a virtual device driver might be necessary if the device has strict interrupt service latency requirements. A virtual device driver/physical device driver (VDD/PDD) pair improves the interrupt service latency for an option adapter or device.

---

* Trademark of the IBM Corporation

Chapter 2. Virtual Device Driver Architecture and Operations

The architecture and structure of a device driver differs considerably, depending on whether the device driver is physical or virtual. A physical device driver is considered a *true* device driver in the sense that it has a unique and rigid file structure, and interfaces directly with the hardware. A virtual device driver is essentially a dynamic link library, and generally does not interface directly with the hardware. Instead, a virtual device driver is responsible for presenting a virtual copy of a hardware resource to a DOS Session, and for coordinating physical access to that resource.

Virtual Device Driver Architecture

Virtual device drivers manage I/O ports, device memory, and ROM BIOS services for the devices they virtualize. For hardware compatibility purposes, the virtual device driver should behave as much as is possible like the physical hardware. Otherwise, incompatible emulation could result. In the case of system drivers, the virtual device driver should use a physical device driver to manipulate hardware, wherever possible.

A virtual device driver is a 32-bit EXE file that can contain some, all, or none of the following types of objects:

- Initialization code
- Initialization data
- Swappable global code.

A virtual device driver must have at least one object of this type:

- Swappable global data
- Swappable instance data
- Resident global code
- Resident global data
- Resident instance data.

The resident objects must be used for code and data that must be accessible at physical hardware interrupt time, that is, when a physical device driver calls the virtual device driver. A virtual device driver that does not interact with a physical device driver may not need any resident objects.

Virtual device drivers conform to the EXE32 Load Module format. A typical virtual device driver has a shared (global) code object, a shared (global) data object, and a private (instance) data object. It is possible for a virtual device driver to have multiple objects of each type. The global code and data objects are addressable in all process contexts. As with instance data for dynamic link libraries, the instance data object is per-DOS Session data; that is, the same address in each DOS Session context is initialized to the same initial state, but backed by different physical memory.

Run-time allocated memory can be allocated either swappable or fixed. In addition, run-time allocated memory can also be:

- Private (per-DOS Session), which is allocated out of the DOS Session private memory area. This is the preferred method, and is suitable when memory is used only in the context of owning the DOS Session.
- Shared (global), which is allocated from the system area. This should be used only when the allocation must be created, modified, or freed in an arbitrary process context.

Virtual Device Driver Operations

Virtual device drivers are loaded and initialized before the Shell is started but after all the physical device drivers have been loaded and initialized.

Virtual Device Driver Loading

After the physical device drivers have been loaded, the DOS Session Manager is initialized (at OS/2 system initialization time), which then initializes the 8086 Emulation component, and loads and initializes the virtual device drivers. An initial V86 mode memory map is constructed at this time for later use at DOS Session creation.

The DOS Session Manager calls the loader to load virtual device drivers. Base virtual device drivers are loaded first, followed by installable virtual device drivers. *Base* virtual device drivers are those that must be present for Multiple DOS Sessions to function. *Installable* virtual device drivers are those specified in the DEVICE= lines in CONFIG.SYS. The Multiple DOS Sessions environment is still operable if any or all of these installable drivers fail to load. The one exception to this rule is the virtual Video device driver; it is a base virtual device driver that is loaded in CONFIG.SYS.

The global code and data objects are loaded into the system area. This is required so that a physical device driver can pass along a hardware interrupt by calling a virtual device driver at interrupt time, regardless of the current process context.

The instance data objects are loaded into the address space between 1MB and 4MB for that session. This is the private address space managed by the DOS Session. The loader and DOS Session Manager cause this allocation to occur so that the full *virtual* state of the DOS Session is contained below the 4MB line, thus allowing a single Page Directory Entry (PDE) to map a DOS Session context. This has two benefits:

- Context-switching DOS Sessions are fast, because only a single PDE needs to be edited.
- It is simple to maintain an alias region in the system area for each DOS Session. This alias region is a 4MB-aligned, 4MB region of linear address space, whose base is called a *DOS Session Handle*, which can be used in a base register to manipulate the instance data of that DOS Session. This is an important feature as it allows virtual device drivers to quickly access the instance data of any DOS Session.

The virtual device driver entry point is called after the virtual device driver is loaded. The device driver returns a non-zero value to indicate a successful load, and returns 0 to indicate an unsuccessful load.

Virtual Device Driver Initialization

After a virtual device driver is loaded, it performs the following operations, as appropriate:

- Verifies presence of corresponding hardware
- Establishes communication with corresponding physical device driver
- Reserves regions of linear memory containing device ROM and RAM
- Saves the initial physical device state for initializing the virtual device state upon DOS Session creation
- Set hooks for various DOS Session events (creation, termination, foreground/background switch). See "VDHInstallUserHook" on page 5-70 for a complete list of user hooks.

Virtual device drivers are initialized at Ring 0. For this reason, they cannot make Ring 3 API calls, and they interact only with OS/2 2.0 through use of the Virtual DevHlp services.

Installing User Hooks: Each virtual device driver must use VDHInstallUserHook to install a VDM_CREATE hook. Virtual device drivers that use foreground and background (for example, Video, Keyboard, and Mouse) must also install VDM_FOREGROUND and VDM_BACKGROUND hooks.

Allocating Global Resources: Any global resources are allocated at this time.

Virtual Device Driver Registering for Communications: When virtual device drivers communicate with the kernel, physical device drivers, other virtual device drivers, or the Presentation Manager interface, the Dynamic Link mechanism is used to:

- Export Virtual DevHlp services to virtual device drivers
- Export kernel data to virtual device drivers
- Export services from one virtual device driver to another.

To restrict the modifications required for physical device drivers (which remain 16:16), Virtual DevHlp services are provided so that a virtual device driver can establish communication with a physical device driver. This is no more than an exchange of entry points. At this point, the physical device driver and the virtual device driver can use any protocol desired to communicate. Although a suggested protocol is provided in "Inter-Device Driver Communication" on page 2-4, device drivers are not restricted to this.

DOS Session Creation and VDD Per-DOS Session Initialization

When the user starts a DOS application, the DOS Session Manager notifies the virtual device drivers that have registered creation hooks, and gives them an opportunity to do per-DOS Session initialization. Memory is allocated and mapped into the V86-mode address space, and the DOS Emulation code is initialized. Control is passed to the DOS Emulation kernel to load the shell specified by DOS_SHELL (usually COMMAND.COM) and the application selected by the user.

Virtual device drivers perform the following operations (as appropriate) at DOS Session creation time:

- Initialize the virtual device state
- Initialize the virtual ROM BIOS state
- Map linear regions (reserved at virtual device driver initialization time) to physical or linear memory
- Hook the I/O ports
- Enable or disable the I/O port trapping
- Hook the software interrupts
- Allocate per-DOS Session memory.

DOS Session Screen Switching: The OS/2 Session Manager notifies the DOS Session Manager of screen switches. The DOS Session Manager, in turn, notifies all virtual device drivers that are involved in these events (generally, the Video, Keyboard, and Mouse virtual device drivers). The virtual Video device driver (VVIDEO) takes appropriate steps to map in physical or logical video RAM and disable/enable I/O port trapping for the video hardware. The virtual Keyboard device driver (VKBD) can set flags to control the reaction to keyboard polling behavior.

DOS Session Destruction: When a user exits a DOS Session, the virtual device drivers are notified to clean up any resources owned by that DOS Session (that is, the virtual device driver's DOS Session termination entry points are called by the DOS Session Manager). Memory is released and the process is terminated, as is done with a normal OS/2 process. DOS Sessions can also be terminated by a virtual device driver if unsupportable behavior is detected, or by the DOS Emulation component (if it is corrupted or is otherwise unable to function).

Local and Global Information Regions

Most of the information available through information segments can be obtained by a virtual device driver using the Virtual DevHlp service, VDHQuerySysValue.

Virtual DevHlp (VDH) Services

A set of Virtual DevHlp (VDH) services is used to provide an abstract but efficient means for virtual device drivers to interact with DOS Sessions and the OS/2 kernel. These VDH services have the following characteristics:

- They are available through dynamic linking.
- They use the 32-bit calling conventions found in the function prototype definition VDHENTRY located in the MVDM.INC include file.
- A return value of 0 usually means the call failed. When 0 is returned, calling VDHGetError returns a detailed error code. If VDHGetError returns 0, then the last call succeeded and 0 was a meaningful return value (not an error). A non-zero return value means the call succeeded.
- All pointer parameters are 0:32 flat pointers.

The Memory Management Virtual DevHlp services are used for:

- Mapping
- Reservation
- Allocation/reallocation/free.

See Chapter 5, "Virtual DevHlp Services" for a list of the page granular memory management services and a description of each Virtual DevHlp.

Inter-Device Driver Communication

A virtual device driver can communicate with a physical device driver by using the Inter-device Driver Communication (IDC) mechanism provided by the OS/2 operating system. A virtual device driver communicates with a physical device driver directly though a callable 16:32 interface, or by using the Virtual DevHlp services that map to File System APIs. Virtual device drivers communicate directly with other virtual device drivers by using a set of the Virtual DevHlp services.

The type and form of communication is defined by the physical or virtual device driver that provides the IDC entry point to be called. These device drivers must be aware of addressability and sensitive to interrupt-time performance. For example, a virtual device driver that is not reflecting hardware interrupts to applications, and is in communication with a physical device driver, should use the Virtual DevHlp services that map to File System APIs to communicate. However, if the virtual device driver must reflect hardware interrupts, it is recommended that the callable 16:32 interface be used because it provides a more sensitive interface in which to communicate.

Virtual Device Driver and Physical Device Driver Interaction

Many virtual device drivers virtualize hardware that generates interrupts. Generally, these drivers must interact with a physical device driver to fully virtualize a device. Virtual DevHlp services are provided for the Open, Close, Read, Write, and IOCtl functions of the File System, which is then routed to the corresponding physical device driver. This is the simplest IDC mechanism for VDD/PDD communication, and is recommended for most IDC requirements.

Where interrupt-time or other high-performance PDD/VDD IDC requirements exist, the VDHOpenPDD service is used to establish communication between a virtual device driver and a physical device driver by exchanging entry points, which use a 32-bit calling convention (refer to the Multiple DOS Sessions-specific equates found in the MVDM.INC include files). At this point, the virtual device driver and physical device driver are free to communicate through whatever private protocol is chosen including exchanging register-based entry points. However, both drivers must agree on a shutdown protocol, that is, what is used to stop the communication if the virtual device driver needs to shutdown.

Virtual Device Driver/Virtual Device Driver Communication

Because dynamic linking is supported between virtual device drivers, multiple virtual device drivers can support inter-virtual device driver communication through dynamic links. Where these device drivers are supplied by multiple parties and it is not required that all of the virtual device drivers in the *set* be present, dynamic linking does not work. The Virtual DevHlp services, VDHRegisterVDD, VDHOpenVDD, VDHRequestVDD, and VDHCloseVDD, are used to overcome this limitation of dynamic linking.

Hardware Emulation and Interrupt Support

The major functions of software interrupt emulation are:

- Hook interrupt service (VDHInstallIntHook)
- Return to DOS Session interrupt handler service (VDHPushInt)
- Hook return service (VDHArmReturnHook).

Hook Interrupt Service (VDHInstallIntHook)

The handler supplied to this service is a post-reflection handler. *Post-reflection handlers* are called after the INT instruction is reflected into the DOS Session interrupt code. A DOS Session breakpoint address is put into the interrupt vector table entry at INIT time, so control returns to the kernel at the end of the DOS Session's interrupt code chain. The virtual device driver interrupt handler chain is then called. The ROM routine address (or whatever was initially in the interrupt vector table), which was replaced by the DOS Session breakpoint, is returned to, if the virtual device driver handlers do not call VDHPopInt.

Return to DOS Session Interrupt Handler Service (VDHPushInt)

VDHPushInt calls the interrupt reflection code to change the DOS Session's execution flow to the DOS Session's interrupt code. This is done by building a return IRET frame on the client's stack with the CS:IP and flags from the kernel stack frame. The DOS Session's CS:IP on the kernel stack is edited with the address from the DOS Session's interrupt vector table.

Hook Return Service (VDHArmReturnHook)

The VDHArmReturnHook service allows the IRET to be hooked after the VDHPushInt service is called. A DOS Session breakpoint is allocated, and the address replaces the client's CS:IP on the return IRET frame. The client's CS:IP is saved.

When building the return IRET frame, the client's stack pointer can wrap around from 0 to 0FFFEH. This does not terminate the DOS Session. If the client's stack pointer is equal to *1* when this function is begun (before the virtual device driver handlers are called), the DOS Session is terminated (to emulate the 386 hardware when it is in real mode).

When the virtual device driver installs an interrupt handler, a DOS Session breakpoint is installed in the interrupt vector table before any DOS Session code hooks. The DOS Session breakpoint is executed after the DOS Session interrupt code chain is executed but before ROM. The virtual device driver interrupt handlers are executed until one returns the *stop chaining* indication, or the end-of-list is reached. ROM code is executed after the last virtual device driver handler is called unless one of these handlers calls VDHPopInt. VDHPopInt emulates the IRET when a V86 code interrupt handler has popped off the IRET frame and restored the DOS Session's CS, IP, and flags to the IRET frame contents. VDHPushInt is used by the software interrupt instruction trap and by the virtual PIC device driver in the last stages of simulating hardware interrupts.

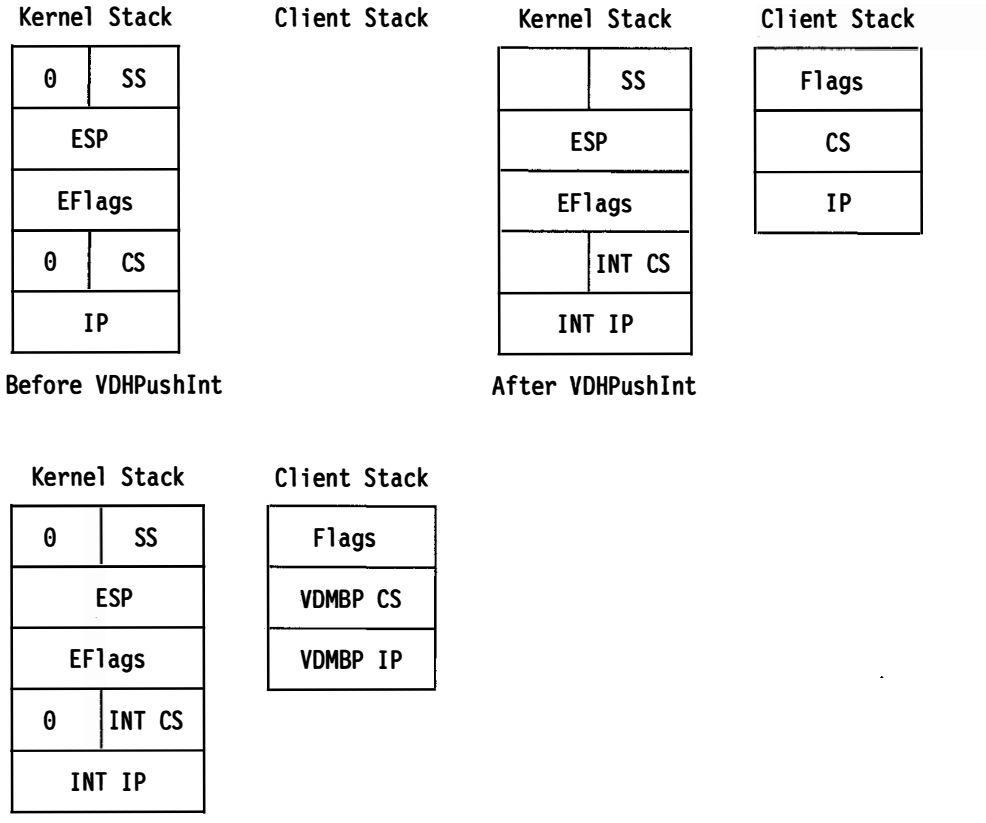


Figure 2-1. VDHPushInt/VDHArmReturnHook Services

Access to Ports Not Trapped by a Virtual Device Driver: In order to support DOS applications that use hardware that does not have virtual device drivers, a mechanism is provided for DOS applications to access ports on those devices. This is done by having the default handler for each I/O port turn *off* trapping for the port in question. All accesses after the first one go straight through to the hardware. This is done only for ports that are not trapped by a virtual device driver.

Global and Local Context Hooks: *Global context hooks* are a mechanism to delay work until the CPU is at task time (as opposed to interrupt time). If a global context hook is set at task time, the handler is called immediately. *Local context hooks* are a mechanism to delay work until the CPU is at task-time in the context of a specific process. When a virtual device driver sets a local context hook, that hook is guaranteed to execute before any user code (that is, code executing in V86 mode).

Interrupts Supported by Multiple DOS Sessions

The following interrupts are supported by multiple DOS Sessions:

- BIOS hardware interrupt support
- BIOS software interrupt support
- DOS software interrupt support.

BIOS Hardware Interrupt Support

In previous versions of the OS/2 operating system, DOS applications were prevented from hooking hardware interrupts when the IRQ level was owned by an OS/2 device driver. A DOS application that violated this rule was terminated. In OS/2 2.0, this restriction is significantly relaxed. Depending on the support provided by the device's virtual device driver, DOS applications can now hook the hardware interrupt vector. The following list summarizes the support provided by OS/2 2.0 virtual device drivers for each of the hardware interrupt request (IRQ) levels:

IRQ 0 Timer (INT 08H). The INT 08H handler is invoked on hardware interrupts from Channel 0 of the system timer. DOS applications can hook this interrupt. See the description of INT 08H in "BIOS Software Interrupt Support" on page 2-8.

IRQ 1 Keyboard (INT 09H). The INT 09H handler is invoked upon the make or break of every keystroke. DOS applications can hook this interrupt.

IRQ 2 Cascade (Slave) Interrupt Controller. Supports interrupt request levels 8 – 15, described below.

IRQ 3 Serial Port (COM2, COM3, COM4). Supported when the virtual and physical COM device drivers are installed.

IRQ 4 Serial Port (COM1). Supported when the virtual and physical COM device drivers are installed.

IRQ 5 Reserved. A virtual device driver to simulate this hardware interrupt is not provided in OS/2 2.0. See "Reserved IRQ Support in OS/2 2.0" for further information.

IRQ 6 Diskette. In OS/2 2.0, this hardware interrupt is not simulated in DOS Sessions. See the description of INT 0EH in "BIOS Software Interrupt Support" on page 2-8.

IRQ 7 Parallel Port. In OS/2 2.0, this hardware interrupt is not simulated in DOS Sessions.

IRQ 8 Real Time Clock. In OS/2 2.0, this hardware interrupt is not simulated in DOS Sessions. See the description of INT 70H in "BIOS Software Interrupt Support" on page 2-8.

IRQ 9 Redirect Cascade. A virtual device driver to simulate this hardware interrupt is not provided in OS/2 2.0. See "Reserved IRQ Support in OS/2 2.0" for further information.

IRQ 10 Reserved. A virtual device driver to simulate this hardware interrupt is not provided in OS/2 2.0. See "Reserved IRQ Support in OS/2 2.0" for further information.

IRQ 11 Reserved. A virtual device driver to simulate this hardware interrupt is not provided in OS/2. See "Reserved IRQ Support in OS/2 2.0" for further information.

IRQ 12 Auxiliary Device. A virtual device driver to simulate this hardware interrupt is not provided in OS/2. See "Reserved IRQ Support in OS/2 2.0" for further information.

IRQ 13 Math Coprocessor (NPX). NPX exception interrupts on IRQ 13 are reflected into the DOS Session.

IRQ 14 Fixed Disk. In OS/2 2.0, this hardware interrupt is not simulated in DOS Sessions.

IRQ 15 Reserved. A virtual device driver to simulate this hardware interrupt is not provided in OS/2. See "Reserved IRQ Support in OS/2 2.0" for more information.

Reserved IRQ Support in OS/2 2.0: The OS/2 operating system does not provide support for optional devices on the *Reserved* and *Redirect Cascade* interrupt levels in OS/2-mode sessions, or in DOS Sessions. Hooking one of these hardware interrupt request levels in a DOS application has no effect. If there is no physical device driver/virtual device driver pair to service an interrupt, the DOS application's interrupt handler is not invoked, even though the hardware interrupts are generated.

These hardware interrupt request levels can be supported in a DOS Session by a user-installed virtual device driver. However, a user-installed physical device driver is also required to provide this support since there is no mechanism for a virtual device driver to directly service a hardware interrupt.

BIOS Software Interrupt Support

BIOS software interrupt compatibility considerations and restrictions for OS/2 2.0 DOS Sessions are shown below. Notice that only specific compatibility considerations and functional restrictions for users of BIOS in the DOS Session environment of the OS/2 operating system are listed.

02H Non-Maskable Interrupt (NMI). Not reflected to the DOS Sessions; this BIOS interrupt handler is not invoked on NMI.

05H Print Screen. The BIOS INT 05H handler is supported. If the DOS application issues INT 05H, the contents of the video buffer is printed. The Shift+PrtScr keystroke combination (or equivalent) also invokes this function.

08H System Timer. This interrupt is invoked every time the system timer Channel 0 counts down to zero (normally, 18.2 times per second). DOS applications can hook this interrupt. However, in a DOS Session, these Timer interrupts can come in a burst and have variable latency. The INT 08H interrupt handler is emulated by the virtual Timer device driver.

0EH Diskette. The INT 0EH handler is not used to service interrupts generated by the diskette device (IRQ 6). DOS applications should not hook this interrupt.

10H Video. The INT 10H functions are fully supported in DOS Sessions. The following INT 10H functions are emulated by the virtual Video device driver.

- **0EH** Write Teletype (TTY)
- **13H** Write String.

Note: These functions are emulated only if the INT 10H vector is unmodified. If any piece of DOS Session code hooks INT 10H, then emulation occurs only if these functions are passed to the virtual Video device driver's interrupt hook.

13H Disk/Diskette. INT 13H emulation in a DOS Session supports both removable and non-removable media. The following subset of INT 13H functions are supported in DOS Sessions:

- **00H** Reset Diskette
- **01H** Read Status
- **02H** Read Sectors
- **03H** Write Sectors (Diskette only)
- **04H** Verify Sectors
- **05H** Format Track (Diskette only)
- **08H** Get Drive Parameters
- **0AH** Read Long (Fixed Disk only)
- **15H** Read DASD Type
- **16H** Change Status (Diskette only)
- **17H** Set Disk Type (Diskette only)
- **18H** Set Media Type (Diskette only).

14H ASYNC. When the virtual COM device driver is installed, all BIOS INT 14H functions are supported. The INT 14H functions are emulated by the virtual COM device driver to enhance performance:

- **00H** Initialize Port
- **01H** Send Character
- **02H** Receive Character
- **03H** Read Status
- **04H** Extended Initialize
- **05H** Extended Port Control.

15H System Services. The following table lists the system services:

AH	Function	Description
00	Cassette Motor ON	Default. Routed to ROM BIOS.
01	Cassette Motor OFF	Default. Routed to ROM BIOS.
02	Cassette Read	Default. Routed to ROM BIOS.
03	Cassette Write	Default. Routed to ROM BIOS.
0F	Format Periodic INT	Error. On return, CF is set.
4F	Keyboard Intercept	Default. Routed to ROM BIOS.
80	Open Device	Default. Routed to ROM BIOS.
81	Close Device	Default. Routed to ROM BIOS.
82	Program Terminate	Default. Routed to ROM BIOS.
83	Event Wait	Support. Supported by VTIMER.
84	Joystick Support	Default. Routed to ROM BIOS.
85	SysReq Key Pressed	Support. Issued by VKBD. On return, AL=0 (key make), or AL=1 (key break).
86	Wait	Support. Supported by VTIMER.
87	Move Block	Error. Managed by VXMS.
88	Get Extended Memory Size	Error. Managed by VXMS. On return, AX=0 (no Extended Memory).
89	Switch to Protect Mode	Error. On return, AX != 0 (fail switch).
90	Device Wait	Default. Routed to ROM BIOS.
91	Device Post	Default. Routed to ROM BIOS.
C0	Get System Config Parms	Default. Routed to ROM BIOS.
C1	Get EBIOS Data Area	Default. Routed to ROM BIOS.
C2	PS/2 Mouse Functions	Error. On return, CF is set, and AH=01 is an invalid function. Notice that all mouse support is through INT 33H.
C3	Watchdog Timeout	Ignore. On return, CF is clear.
C4	Prog Option Select	Default. Routed to ROM BIOS.

16H Keyboard. Supported directly by the ROM BIOS INT 16H routines.

17H Printer. Supported (emulated) by the virtual Parallel Port device driver.

19H Reboot. Supported. However, this does not operate in the same manner as DOS (the system is not restarted). If the DOS Setting, DOS_STARTUP_DRIVE, is set, the diskette or image re-boots. Otherwise, the DOS Session is terminated.

1AH Time of Day. The virtual CMOS device driver component supports read-only access to the Real Time Clock device. Because of this restriction, the following BIOS INT 1AH functions are not supported in DOS Sessions:

01H Set RTC Count
03H Set RTC Time
05H Set RTC Date
06H Set RTC Alarm
07H Reset RTC Alarm
08H Set RTC Activated Power-on mode
0BH Set System Timer Day Counter

80H Setup Sound Multiplexer.

A DOS application can read the RTC Count, Time, and Date. The INT 1AH functions are not emulated by the virtual CMOS device driver. Instead, the BIOS ROM functions are executed directly. The RT/CMOS hardware is virtualized by the virtual CMOS device driver.

1EH Diskette Parameters. The segment address of the diskette parameters table is stored in the INT 1EH entry.

70H Real Time Clock Interrupt. Interrupts from the RTC (IRQ 8) are not reflected into DOS Sessions. This software interrupt handler is not used for either alarm or periodic interrupts.

DOS Software Interrupt Support

The DOS Emulation component supports the documented aspects of the following DOS features:

20H Program Terminate

21H DOS Function Request. All documented and undocumented INT 21H functions are supported by DOS Emulation. However, the following functions are supported with restrictions:
4410H
4411H
4B05H
5DH
5EH
5FH
64H
69H
6DH
6EH

Note: Function 53H is not supported by DOS Emulation.

22H Terminate Address
23H Ctrl + Break Exit Address
24H Critical Error Handler
25H Absolute Disk Read
26H Absolute Disk Write. A hard error is reported on requests for non-removable media.
27H Terminate but Stay Resident
28H Idle Loop
2FH Multiplex. Time slice yield (AX = 1680H) is supported. When a DOS application issues INT 2FH with AX = 1680H, it offers to yield its time slice. This can be used by DOS applications in busy wait loops (preserves all registers).

Other Software Interrupt Support

The following are other software interrupts supported in the DOS Session environment of OS/2 2.0:

INT 33H - Mouse: Supported. When the OS/2 virtual Mouse driver is installed, the full range of INT 33H functions, as defined in the *Microsoft Mouse Programmer's Reference Guide*, are available.

INT 67H - LIM Expanded Memory Manager (EMM): Supported. When the OS/2 virtual Expanded Memory Manager device driver is installed, the Lotus**/Intel** /Microsoft** (LIM) EMM V4.0 INT 67H functions, as defined in the *Lotus/Intel/Microsoft Expanded Memory Specification Version 4.0*, are available.

** Lotus is a trademark of the Lotus Development Corporation

** Intel is a trademark of the Intel Corporation

** Microsoft is a trademark of the Microsoft Corporation

Part 2. OS/2 2.0 Virtual Device Drivers

Chapter 3. Base Virtual Device Drivers

The following list contains all the base virtual device drivers included with OS/2 2.0:

- Virtual BIOS device driver (VBIOS)
- Virtual CMOS device driver (VCMOS)
- Virtual Direct Memory Access device driver (VDMA)
- Virtual Diskette device driver (VFLPY)
- Virtual Fixed Disk device driver (VDSK)
- Virtual Keyboard device driver (VKBD)
- Virtual Numeric Coprocessor device driver (VNPX)
- Virtual Parallel Port device driver (VLPT)
- Virtual Programmable Interrupt Controller device driver (VPIC)
- Virtual Timer device driver (VTIMER).

Virtual BIOS Device Driver

The virtual BIOS device driver (VBIOS) contains the mechanism to initialize the Interrupt Vector table, BIOS data area, extended BIOS data area (on a PS/2*), adapter ROM area, and the system ROM. It also provides virtual access to Programmable Option Select (POS) registers (on a PS/2) and supports some of the DOS Settings.

DOS Session Creation

When a DOS Session is created, the virtual BIOS device driver is responsible for allocating and initializing the following areas in the DOS Session's address space with the respective values from physical memory:

- Interrupt Vector Table
- BIOS data area
- DOS communications area
- Extended BIOS data area (on a PS/2).

The virtual BIOS device driver is also responsible for mapping the system ROM and adapter ROM into the address space of the DOS Session. The per-DOS Session copy of the POS registers is initialized at this time.

DOS Settings

The virtual BIOS device driver is responsible for Include/Exclude region support. The DOS Setting, MEM_INCLUDE_REGIONS, gives the user an option to give more space to EMM/XMS. If the user knows that in a particular DOS Session an adapter will not be used, then the linear region taken by that adapter can be given to EMM and XMS through MEM_INCLUDE_REGIONS. MEM_EXCLUDE_REGIONS gives the user an option to restrict EMM and XMS from claiming a particular region. Both these options are provided through DOS Settings. These DOS Settings are only applicable at DOS Session creation time.

Adapter ROM

At system initialization, the virtual BIOS device driver searches the region between the top of the DOS user space (RMSIZE) and 1MB to find all of the adapter ROM. Adapter ROMs that include the ROM signature will be found. In a normal DOS Session creation (without any Include/Exclude properties specified), these linear addresses are mapped to corresponding physical ROM.

All the ranges specified in the DOS Setting, MEM_INCLUDE_REGIONS, are taken out of the list used for mapping the linear addresses to corresponding physical ROM. As a result, MEM_INCLUDE_REGIONS is not mapped to physical ROM and it remains available to EMM or XMS.

For MEM_EXCLUDE_REGIONS, the virtual BIOS device driver reserves those regions and maps them to their corresponding physical addresses. This allows the user the ability to map adapter ROM that does not contain the ROM signature and adapter RAM.

* Trademark of the IBM Corporation

Virtual CMOS Device Drivers

The virtual CMOS device driver consists of virtual support for the CMOS battery backed-up RAM, the Real Time Clock (RTC) and the Non-Maskable Interrupt (NMI) disable logic. It provides virtual access to the address and data latches through virtual I/O ports. This component allows the majority of existing DOS applications that access its ports by direct access, or by normal BIOS calls, to operate without change.

Supported Functions

The following functions are supported by the virtual CMOS device driver:

- CMOS memory access
- I/O port support
- NMI-disable
- Read Time of Day and Date
- Real Time Clock and interrupt access.

CMOS Memory Access: CMOS Memory Access provides support for passive reads and writes to virtual CMOS memory. The CMOS portion of the CMOS/RTC can be read or written. The virtual CMOS memory is initialized to the contents of the physical CMOS memory (at the time of DOS Session initialization) through the physical Clock device driver, which is the physical device driver of the virtual CMOS component. Values are written to the virtual CMOS memory in a buffer local to the DOS Session. Unlike the physical CMOS memory, however, the contents of the virtual CMOS memory are lost when the DOS Session is terminated.

I/O Port Support: The virtual CMOS device driver component monitors all accesses to its two DOS Session I/O ports. The two ports are a write-only address latch and a read/write data latch. The address latch port has two functions, NMI Disable, and CMOS/RTC Device Address Selection. The data latch is a register for holding a byte being transferred to, or from, the CMOS/RTC device.

NMI-Disable: The NMI-Disable portion of the the address latch can be set or reset by a DOS application. However, changes to enable or disable NMI are otherwise ignored.

Read Time of Day and Date: This function presents the correct seconds, minutes, and hours to an application reading the appropriate locations. This function also provides atomic access to all values to avoid reading the values during time transition.

Real Time Clock and Interrupt Access: The Real Time Clock consists of a time-of-day clock, an alarm interrupt, and a periodic interrupt. Accesses to the Real Time Clock to change the time of day, the timing mode, or to set an alarm or periodic interrupt are disallowed. The CMOS/RTC registers related to the Real Time Clock are supported for read-only access. Because interrupts can only be supported through write access to the ports, Real Time Clock interrupts are not supported.

BIOS Support

The virtual CMOS device driver component does not emulate BIOS functions. Instead, it allows the existing BIOS functions to be used without change.

Unsupported Functions

From the perspective of the DOS Session, the virtual CMOS device driver component supports access to I/O address space only. Therefore, the following functions are not supported:

- Set Time of Day and Date
- Set Time of Day Alarm
- Set/Enable Periodic Interrupt
- Enable Update Interrupt.

Virtual Direct Memory Access Device Driver

The virtual Direct Memory Access device driver (VDMA) provides per-DOS Session per-Direct Memory Access (DMA) channel virtualization. In addition, the virtual DMA device driver provides the following:

- Serializes the DOS Session's DMA requests on a per-DMA channel basis (but allows simultaneous requests on different DMA channels)
- Converts the DOS Session's linear addresses to physical addresses, and handles all the data movement required
- Handles all the restrictions of a DMA chip.
- Supports Enhanced DMA operations and commands of a PS/2. Optimizes DMA virtualization on PS/2s by using the Enhanced modes.

In addition to the above features, the virtual DMA device driver also synchronizes the operations with other virtual device drivers registered on different DMA channels. A virtual device driver for a piece of hardware, which uses a DMA channel, registers with the virtual DMA device driver. The virtual device driver informs these virtual device drivers before starting and stopping the DMA operation on their channels.

Virtual Diskette Device Driver

The virtual Diskette device driver (VFLPY) intercepts applications that try to access the diskette drive directly or issue INT 13H calls. This virtual device driver also serializes and coordinates I/O operations between Multiple DOS Sessions.

Obtaining Physical Diskette Hardware Ownership

When the system first starts up, the physical Diskette device driver owns the physical diskette hardware. All diskette accesses from the OS/2 applications go through the file system or the physical Diskette device driver, and are eventually serialized into requests. Initially, the virtual Fixed Disk device driver does not own the diskette controller, and it hooks all the diskette ports. It requests ownership from the physical Diskette device driver when the DOS Session accesses any diskette port, or when the virtual DMA device driver informs the virtual Fixed Disk device driver that the diskette DMA channel has been unmasked. If the immediate ownership request is not successful, the DOS Session is blocked until the physical Diskette device driver calls back and grants the ownership.

Releasing Physical Diskette Hardware Ownership

Once the virtual Diskette device driver has obtained the diskette ownership, it does not release ownership until the physical Diskette device driver requests it, or the diskette motor is turned *off*.

Physical Diskette Interrupt Routing

Physical diskette interrupts are always routed to the physical device driver by the interrupt manager. If the virtual Diskette device driver has the exclusive hardware ownership when the interrupt happens, the physical Diskette device driver routes the event to the virtual Diskette device driver, which simulates the interrupt to the DOS Session.

Virtual Fixed Disk Device Driver

The virtual Fixed Disk device driver (VDSK) provides virtual disk support for the Multiple DOS Sessions environment and supports access to the disk through BIOS (INT 13H) calls. Since BIOS accesses the hardware ports directly, the virtual Fixed Disk device driver component overrides the BIOS by emulating INT 13H directly for hard files. The virtual Fixed Disk device driver component does not provide I/O port level access to the disk controllers. Access to hardware ports that are supported for the diskette drive, and BIOS INT 13H calls, are not emulated; instead, they are run from the actual BIOS. INT 13H support includes:

AH = 00H Reset diskette system
AH = 01H Status of disk system
AH = 02H Read sectors into memory (both diskette and hard disk)
AH = 03H Write sectors from memory (diskette only)
AH = 04H Verify sectors (both diskette and hard disk)
AH = 05H Format track (diskette only)
AH = 08H Get current drive parameters (both diskette and hard disk)
AH = 0CH Seek
AH = 15H Get disk type (both diskette and hard disk)
AH = 16H Change of disk status (diskette only)
AH = 17H Set disk type (diskette only)
AH = 18H Set media type for format (diskette only).

The INT 13H emulator is also responsible for maintaining the diskette and hard disk areas of the ROM BIOS data area.

Typical steps of an INT 13H access are listed below:

- A DOS application accesses the disk through the INT 13H interface. The INT 13H call is trapped by the virtual Fixed Disk device driver.
- A request packet is built and sent to the physical Fixed Disk device driver. Then, the DOS Session is blocked waiting for the request to finish.
- Upon receiving the request packet, if the disk is currently busy, the physical Fixed Disk device driver queues the request.
- When the request is completed, the physical device driver notifies virtual device driver, which unblocks the DOS Session.

Because the virtual Fixed Disk device driver component does not support register level accesses of the disk controller, all accesses to its registers are trapped and ignored.

Note: Timing related copy protection can fail because real time access is not guaranteed. Sector WRITE and FORMAT requests are not supported for hard disk. Sector WRITE and FORMAT operations are supported for diskette.

Virtual Keyboard Device Driver

The virtual Keyboard device driver (VKBD) consists of virtual support for the keyboard, and allows keystrokes to be passed from the keyboard to the DOS Session and text to be pasted from another application into the DOS Session as keystrokes. This virtual device driver also allows existing DOS applications, that access the keyboard, to operate without change. Such applications can access the keyboard through BIOS calls, access to the BIOS data area, or monitoring of the keyboard interrupt, and reading keystroke data directly from the I/O ports.

The virtual Keyboard device driver addresses the following issues:

- INT 21H Access. DOS applications can access the keyboard by opening the CON: device, or by default standard input access.
- BIOS Access. DOS applications can access the keyboard by BIOS INT 09H and INT 16H.
- I/O Port Access. DOS applications can access the keyboard by reading and writing I/O ports 60H and 64H. This access is not restricted to reading scan codes, and can include other types of access (such as LED control).
- Pasting. Data can be *pasted* in from other OS/2 applications.
- Internationalization. Keystrokes are translated according to the current keyboard translation table and code page.
- Idle Detection. DOS applications must not waste excessive CPU time when in input peek loops.

Levels of I/O Support

The various levels of support for the virtual Keyboard device driver are as follows:

- BIOS support
- CON device driver support
- DOS support
- Idle detection support
- I/O port support
- Pasting support.

BIOS Support: Since the BIOS INT 16H service only references the BIOS data area for keystrokes, the BIOS continues to provide this service.

The BIOS INT 09H interrupt service routine, however, is emulated within the virtual Keyboard device driver. When the INT 09H vector is invoked, the virtual Keyboard device driver passes the character to the physical device driver, which translates the character according to the current translate table.

It is possible for an application or a DOS memory resident program to *hook* the INT 09H vector. Often, such applications simply monitor keystrokes, and pass them on to the previously hooked interrupt service routine. In this case, the virtual Keyboard device driver emulation code still runs and translates scan codes according to the current translation table. Otherwise, if an application, which has hooked INT 09H, fails to pass the scan code on, then the scan code is not translated, or placed into the BIOS data area by the virtual Keyboard device driver.

CON Device Driver Support: This support consists of allowing the virtual Console device driver to be replaced. Although the default virtual Console device driver restricts keyboard access to BIOS calls, user-supplied drivers can access the keyboard through I/O ports or through the BIOS data area.

DOS Support: This support consists of keyboard access through the various INT 21H functions. In particular, this is support for access from the DOS Emulation Component. For compatibility, DOS calls the virtual Console device driver to read keystrokes. This allows the virtual Console device driver to be replaced without ill effect.

Idle Detection Support: DOS applications waste CPU time in a multi-tasking system when they peek for input in a loop. Idle conditions are detected when:

- Application is polling the keyboard using INT 16H in a loop
- DOS is polling in an input loop
- Repeated INT 28H calls from a DOS application
- INT 2FH, AX=1680H yield calls
- DOS INT 21H, service 0BH.

Two services are available to device drivers to assist and for handling, "VDHReportPeek" on page 5-133, and "VDHWakeIdle" on page 5-159.

I/O Port Support: The virtual Keyboard device driver is responsible for supporting programs that access the keyboard ports directly. These ports provide access to data ports on the hardware that controls I/O to the keyboard. Primarily, this support is provided for applications that simply read the scan code directly from the port. However, other functions are possible such as setting the key repeat rate. Repeat rates affect the entire system; per-DOS Session repeat rates are not supported. Extended scanner functions, such as providing keyboard pre-controller translation, are not supported; such commands are ignored.

Pasting Support: The virtual Keyboard device driver supports two levels of pasting from the clipboard into a DOS Session, fast pasting and slow pasting. In *fast pasting* support, characters from the clipboard are put in the BIOS buffer directly. For those DOS Session applications, which get their key strokes from the BIOS key buffer (through INT 16H), fast pasting is adequate. For those DOS Session applications, which hook INT 09H or INT 15H (AH=4FH), key scan sequences are expected, which must be generated by the slow pasting mechanism.

In *slow pasting*, every character from the clipboard is reverse translated into scan sequence and can involve generating make-and-break scans of not only the character itself, but also the make and break of the shift keys in order to adjust the upper, lower, or control case of the character. It can even involve generating *alt-numpad* scan sequences for those characters, which cannot be directly typed on the keyboard (such as the graphics, or international characters).

Virtual Numeric Coprocessor Device Driver

The virtual Numeric Coprocessor device driver (VNPX) provides virtual coprocessor (80x87) support for multiple DOS Sessions. The virtual NPX device driver coordinates exception and interrupt reflection for the real coprocessor. Machines with the 80386 B1 processor, or without a coprocessor, do not have coprocessor support.

Virtual Parallel Port Device Driver

The virtual Parallel Port device driver (VLPT) consists of virtual support for direct hardware access to the three parallel ports (LPT1, LPT2, and LPT3), and for the printer-related BIOS entry points (Printer I/O INT 17H, Print Screen INT 05H). Existing DOS applications that access the parallel ports, either by direct access to the ports or by printer BIOS, operate without change. The virtual Parallel Port device driver provides the virtualization to allow multiple DOS applications to access virtual printer hardware simultaneously. This virtual device driver, in conjunction with the spooler, attempts to prevent multiple DOS applications that transmit data by using BIOS functions or direct hardware access, from intermixing output at the parallel port.

The virtual Parallel Port device driver consists of several sections:

- BIOS INT 17H Emulation
- BIOS INT 05H Emulation
- Print Close Processing
- Direct I/O Access.

BIOS INT 17H Emulation

One method DOS applications can use to send data to a printer is to use BIOS interrupt 17H. The virtual Parallel Port device driver emulates BIOS INT 17H and passes these requests through the File System to the physical Parallel Port device driver by using the VDHOpen, VDHWrite, and VDHClose system services. These functions are not passed to BIOS for processing.

The BIOS INT 17H emulation supports three functions:

- Print Character. The Print Character function buffers single character requests in the virtual device driver buffer associated with each device in a DOS Session. When the buffer becomes full, its contents are sent to the physical Parallel Port device driver by using VDHWrite. If the device is not open, the virtual Parallel Port device driver issues VDHOpen to obtain a file handle and start a printer data stream.
- Initialize. The virtual Parallel Port device driver buffer is sent to the physical Parallel Port device driver by using VDHWrite. The printer data stream is closed by using VDHClose.

  When the spooler is enabled, closing the data stream closes the spool file. Any subsequent printing will open a new spool file. The beginning of the spool file contains escape sequences to reset the printer to its *power on* defaults. When the spooler is disabled, no initialization of the device takes place.
- Read Status. The Read Status function returns the contents of the virtual device status register.

See "Printer Close Processing" on page 3-12 for the possible methods of closing a printer data stream opened with BIOS INT 17H.

BIOS INT 05H Emulation

The method used to send data from the display to the printer (LPT1) is to press the PrtScr key. DOS applications can also initiate a Print Screen function by issuing a call to a BIOS INT 05H function. The virtual Parallel Port device driver passes this request to BIOS for processing. The virtual device driver performs some special processing to prevent the INT 05H request from intermixing with an INT 17H print request to the same device. Also, two BIOS INT 05H requests cannot be performed simultaneously from within the same DOS Session. The second print screen request is ignored.

Printer Close Processing

BIOS printer interrupts operate on a single character basis, which can have a negative impact on printer and system performance. For this reason, the virtual Parallel Port device driver buffers BIOS print requests. As the buffer becomes full, it is sent to the File System to be spooled and printed. The size of the application's print request is rarely an even multiple of the virtual Parallel Port device driver buffer size. This causes the virtual device driver to provide methods of flushing the remaining characters in its buffer.

A virtual Parallel Port device driver's BIOS INT 17H buffers are flushed in one of four ways:

- DOS Settings PRINT_TIMEOUT automatic close feature
- Ctrl + Alt + PrtScr key sequence (VDHPrintClose)
- Application and DOS Session termination.

DOS Settings PRINT_TIMEOUT: The user can set the DOS Setting PRINT_TIMEOUT to specify the length of time in seconds that the virtual Parallel Port device driver will wait between occurrences of INT 17H before flushing its buffer. A setting of *0* specifies to wait indefinitely, that is, to never flush the buffer using the timeout mechanism. The PRINT_TIMEOUT mechanism flushes the buffer and closes the data stream automatically each time the DOS application stops printing by using INT 17H for the time specified.

Ctrl + Alt + PrtScr Key Sequence: After the DOS application has printed the entire file (issued INT 17H), the user can simultaneously press the Ctrl, Alt, and PrtScr keys, which alerts the virtual Parallel Port device driver to flush the remaining characters in its buffer. The virtual Keyboard device driver calls VDHPrintClose (an exported virtual Parallel Port device driver entry point) to alert the virtual Parallel Port device driver that the key sequence has been pressed, and to flush the buffer.

Application and DOS Session Termination: The virtual Parallel Port device driver is notified when the application terminates or the DOS Session is closed, and will flush the remaining characters in its buffer.

Direct I/O Access

The virtual Parallel Port device driver, at its initialization, registers an Inter-device Driver Communication (IDC) with the physical Parallel Port device driver. The virtual Parallel Port device driver requests exclusive access to the parallel port hardware through the VDD/PDD IDC when a DOS application interfaces with the hardware directly. Data transmitted using the direct I/O access method will bypass the spooler. At completion of the exclusive access to the parallel port hardware, the port will be initialized to reset it to its initial state.

Interrupt-Driven Data Transfers

The virtual Parallel Port device driver does not reflect virtual hardware interrupts to applications executing in a DOS Session. Attempts by a DOS application to set bit 4 (IRQ EN) of the parallel port device control register are ignored.

Bidirectional Data Transfers

The extended mode of the PS/2 parallel port allows the port to be used as a bidirectional 8-bit parallel interface. This capability is supported by the virtual Parallel Port device driver when DOS applications access the parallel port hardware directly.

Virtual Programmable Interrupt Controller (PIC) Device Driver

The virtual PIC device driver (VPIC) is responsible for the virtualization of the 8259 Programmable Interrupt Controller (PIC) for the Multiple DOS Sessions environment, and for the simulation of interrupts to DOS Sessions. Performance is a major requirement for hardware interrupt simulation. In this book, *hardware interrupt* and *interrupt* have the same meaning, but *simulated interrupt*, *interrupt simulation*, and *hardware interrupt simulation* refer to the virtual PIC device driver sending an interrupt to a DOS Session.

The virtual PIC device driver supports the hardware interrupt-related services needed by virtual device drivers and DOS Sessions. The services include setting handlers to trap EOI and IRET events, simulating interrupts to DOS Sessions, and handling PIC I/O accesses by DOS Sessions. See "Virtual Interrupt Services" on page 5-5 for a list of these Virtual DevHlp services. The virtual PIC device driver maintains a per-DOS Session virtual PIC state so that each DOS Session appears to have its own independent 8259 Programmable Interrupt Controller. Provisions are included to demand page DOS Sessions interrupt handling code. The virtual PIC device driver provides a DOS Session with the following services:

- Virtualization of the 8259 PIC ports:
  - Separate master and slave PICs
  - Mask and unmask IRQ commands
  - Specific and non-specific EOI commands
  - Read IR and IS register commands.
- Hardware interrupts that can be be postponed until task time so that a DOS Session can be paged
- Interrupts from an optional hardware adapter, which does not have a virtual device driver and a physical device driver, can be simulated to a DOS application.

The virtual PIC device driver provides virtual device drivers with:

- A high performance mechanism to send hardware interrupts to DOS Sessions:
  - Support for more than one virtual device driver to share an interrupt level
  - Support for a virtual device driver to regain control when a DOS Session issues EOI commands, or when a DOS Session executes the IRET at the end of the interrupt routine.
- A mechanism for virtual device drivers to send a virtual EOI to the virtual VPIC device driver
- A way to block until an interrupt is simulated
- A mechanism to query a DOS Session's virtual IRQ state or status:
  - Masked or unmasked state (IMR)
  - Waiting to be sent to a DOS Session state (IRR)
  - Executing DOS Session's interrupt handler code state (ISR)
  - Pending interrupt routine IRET state.

Hardware interrupt simulation is closely tied to the PIC virtualization. Virtual device drivers need a fast mechanism to call a DOS Session's real-mode interrupt handler for a hardware interrupt. Because paging of DOS Sessions (and therefore DOS Session interrupt handlers) is required, simulated interrupts are delayed until the task-time context of the target DOS Session.

Virtual Timer Device Driver

The virtual Timer device driver (VTIMER) provides virtual timer support for the Multiple DOS Sessions environment. This virtual Timer device driver emulates the Intel** 8254A Programmable Interval Timer.

DOS Support

The virtual Timer device driver provides the following:

- Virtualization of timer ports to allow reprogramming of the interrupt rate and the speaker tone frequency
- Distribution of timer ticks to DOS Sessions
- Maintenance of the timer tick count in the BIOS data area
- Serialization of Timer 0 and Timer 2 between the virtual Timer device driver and the physical Time device driver.

** Intel is a trademark of Intel Corporation

Chapter 4. Installable Virtual Device Drivers

The following list contains all the installable virtual device drivers included with OS/2 2.0:

- Virtual CDROM device driver (VCDROM)
- Virtual COM device driver (VCOM)
- Virtual DOS Protect Mode Extender device driver (VDPX)
- Virtual DOS Protect Mode Interface device driver (VDPMI)
- Virtual Expanded Memory Specification device driver (VEMM)
- Virtual Extended Memory Specification device driver (VXMS)
- Virtual Mouse device driver (VMOUSE)
- Virtual Touch device driver (VTOUCH)
- Virtual Video device drivers (VCGA, VEGA, VMONO, VVGA, VXGA, V8514A)

Virtual CDROM Device Driver

The virtual CDROM device driver (VCDROM) enables audio support for CDROM applications running in DOS Sessions of OS/2 2.0. Under DOS, audio and other IOCtl support is provided through the Pass-Through function of the CDROM File System driver, MSCDEX. DOS applications build a device driver IOCtl request packet and then send it to MSCDEX to be forwarded to the virtual CDROM device driver for service.

The virtual CDROM device driver provides two distinct features that are necessary to support DOS CDROM applications. First, the virtual CDROM device driver emulates the presence of MSCDEX by providing the services that CDROM applications require for installation (for example, the version of MSCDEX, the number of CDROM devices that are present in the system, and the drive letters that correspond to the CDROM devices.)

The second, and major, function of the virtual CDROM device driver is to translate the DOS-style IOCtls into requests that the physical CDROM device driver can understand. The virtual and physical CDROM device drivers provide roughly the same device services but through different IOCtl interfaces. The DOS CDROM interface is available through the IOCtl input and output device driver request packets, while the physical CDROM device driver provides its services through the generic IOCtl interface. The virtual CDROM device driver maps the DOS IOCtls to their corresponding physical device driver IOCtls, and then passes them on to the physical CDROM device driver. On return from the physical CDROM device driver, the virtual CDROM device driver must also map the return information and the return codes back into the DOS-style environment.

The virtual CDROM device driver provides only audio IOCtl support not a full emulation of MSCDEX. Most DOS CDROM applications use the standard DOS interface for File System services, and use the MSCDEX interface only for audio IOCtl services. These applications are fully supported. However, if an application calls MSCDEX directly for File System services, it will not run in a DOS Session of OS/2 2.0.

Virtual COM Device Driver

The virtual COM device driver (VCOM) consists of virtual support for the serial communication I/O ports, and for the serial channel-related BIOS entry points. It provides support in each DOS Session for up to four COM ports. In addition, this component allows existing DOS applications, which access communications channels by direct access to the I/O ports or by normal BIOS calls, to operate without change.

The virtual COM device driver only supports access to communication channels that are physically present on a given system. This does not include support for accessing communication devices that can be redirected by network software. From the perspective of the DOS Session, the virtual COM device driver supports access through:

- BIOS
- I/O address space.

Support Summary

The virtual COM device driver directly supports invocations of BIOS INT 14H, instead of allowing the BIOS to access the ports directly. However, direct access to the virtual I/O ports is also supported. Direct access to I/O ports results in emulation of a communication controller with the INS8250 UART. When referenced through the virtual I/O ports, the virtual COM device driver also provides all support for virtual interrupt indication. The hardware appears to the DOS Session as a serial interface supported by standard expansion bus serial ports on a PS/2* computer.

Interrupt Service

The virtual COM device driver provides full support for the virtualization of communication channel interrupts, and follows the same logic as the actual hardware.

* Trademark of the IBM Corporation

Virtual DOS Protect Mode Extender Device Driver

The virtual DOS Protect Mode Extender device driver (VDPX) provides address translation from protect mode to V86 mode for DOS Protect Mode Interface (DPMI) applications. This translation is necessary because DPMI applications run in protect mode but issue interrupt requests, which can be handled in V86 mode, to perform various system services. When running in protect mode, the application is using selectors instead of segments, therefore, any data buffers in the protect-mode memory address space (above 1MB) must be copied into a region of memory within the V86 address space.

The virtual DPX device driver registers the DOS Setting, DPMI_DOS_API, which controls whether the virtual DPX device driver is active or not. If the user selects:

Enabled The virtual DPX device driver always translates the interrupts it supports from protect mode to V86 mode.

Auto The application must first issue an INT 2FH in protect mode to begin the translation.

Disabled The virtual DPX device driver does not perform any address translation.

DPMI applications can provide their own translation services, therefore, it might be unnecessary for the virtual DPX device driver to perform any address translation.

The virtual DPX device driver translates various interrupts from protect mode to V86 mode, including INT 10H, 15H, 21H, 25H, 26H, 2AH, 33H, and 5CH. The virtual DPX device driver translates most of the commonly used functions for these interrupts. After the virtual device driver translates the interrupt, the V86-mode interrupt service routine (or the virtual device driver, if it has hooked the interrupt) handles the interrupt request. After issuing the interrupt in V86 mode, the virtual DPX device driver insures that any data returned from the V86 service is copied into the data area originally passed in by the DPMI application. The registers and flags returned by the V86-mode handler (other than mode-sensitive registers such as Segment registers) are returned to the DPMI application.

Virtual DOS Protect Mode Interface Device Driver

The virtual DOS Protect Mode Interface device driver (VDPMI) provides Version 0.9 DPMI support for applications in a DOS session. DPMI applications run in protect mode instead of V86 mode. The virtual DPMI device driver provides the necessary support to allow DPMI applications to run under OS/2 2.0. The virtual DPMI device driver supports the DPMI Memory Limit DOS Setting, which allows the user to specify the how much protect-mode memory should be available to the DOS Session.

The virtual DPMI device driver supports INT 31H in protect mode, which is issued by DPMI applications to request a DPMI service. The virtual DPMI device driver keeps track of the resources used by the application, such as memory, selector, interrupt, and debug watchpoint management. This virtual device driver also supports the mechanisms necessary to allow DPMI applications to switch the DOS Session in and out of V86 mode (translation services). When a DPMI application terminates, the virtual DPMI device driver ensures that all of the DPMI application resources are freed.

If a DPMI application issues an interrupt in protect mode and there is no protect-mode interrupt handler, the request is reissued in V86 mode. Most virtual device drivers hook only interrupt requests in V86 mode, so they are not called until the interrupt is reflected to V86 mode (unless the virtual device driver has issued VDHSetVPMIntVector). However, it is up to the application or the virtual DOS Protected Mode Extender device driver (VDPX) to do any address translation that is necessary, such as converting the selector address to a segment address. See "Virtual DOS Protect Mode Extender Device Driver" on page 4-4 for further information on address translation.

Virtual Expanded Memory Specification Device Driver

The LIM Expanded Memory Specification (EMS) Version 4.0 provides a standard interface for the use of expanded memory on 8086 family computers. This specification offers up to 32MB of expanded memory divided into as many as 255 expanded memory objects. Regions of these objects can be mapped at 8086 addresses (below 1MB), thus allowing DOS applications access to large memory allocations. However, the memory that is to be accessed must be explicitly mapped.

Alternate page tables are provided for quick switches among mappings and function calls with remapping. These tables are also used as the means to save and update mappings, or move and exchange memory contents.

The OS/2 virtual Expanded Memory Specification (EMM) device driver:

- Implements all the INT 67H functions in the LIM 4.0 EMS, except those for the DMA registers.
- Provides each DOS Session with a separate EMS emulation. A DOS Session has its own set of expanded objects so that features such as interprocess communication work as if each DOS Session was running on a different real 80386. DOS Sessions cannot affect the availability of objects in other DOS Sessions, or access memory in other DOS Sessions.
- Provides support for remapping of conventional memory (below 640KB) for use by programs.
- Provides a configurable limit for the amount of EMS memory available per DOS Session.
- Supports multiple physical to single logical mappings. Different 8086 addresses can map to the same expanded memory object address.

Configurable Defaults

Default values are set for properties in order to be appropriate in most instances. However, users can change these defaults, if desired. The following defaults can be set in CONFIG.SYS:

Default EMS Memory Size

Default limit set by specifying the number of kilobytes of memory desired on the DEVICE= line in CONFIG.SYS. Alternately, the limit can be specified by using a switch, /S=xxx, on the CONFIG.SYS line. For example:

```
device=path\VEMM.SYS 1024
```

or

```
device=path\VEMM.SYS/S=1024
```

Default EMS Low OS Map Region

Size of remappable conventional memory set by using the /L switch. For example:

```
device=path\VEMM.SYS /L=256.
```

EMS High OS Map Region

Size of extra mappable memory (other than the 64KB minimum) set by using the /H switch. For example:

```
device=path\VEMM.SYS /H=32.
```

EMS Frame Location

Default frame position set by using the /F switch. Any value in the DOS Setting list can be chosen as the default. For example:

```
device=path\VEMM.SYS /F=AUTO
```

or

```
device=path\VEMM.SYS/F=C800
```

When devices without virtual device drivers can directly map memory, the Include and Exclude regions are crucial for the virtual EMM device driver. These properties are supplied by another component, and determine which regions can be directly mapped to physical addresses by default when touched by applications.

The Virtual DevHlp Mapping services allow the same master page to be aliased in by successive pages in the V86 address space. In particular, EMM allows an application to map in the same 16KB size range of memory repeatedly (for example, a 64KB page frame could consist of the same four pages mapped 4 times). Since DMA requires contiguous physical memory, I/O that involves aliased pages, which map to the same physical page, must be handled by breaking up DMA requests.

Virtual Extended Memory Specification Device Driver

The Extended Memory Specification (XMS) Version 2.0 provides a standard interface for the use of extended memory on 80286-based and 80386-based machines. The XMS specifications manage three distinct regions of memory:

- High Memory Area (HMA). Extended memory accessible from real mode by enabling the A20 address line. Code can be executed in the HMA, if the A20 line is enabled. The high memory area is exactly 65520 bytes (64KB − 16 bytes) long.
- Extended Memory Blocks (EMBs). Blocks of extended memory accessible by way of XMM function calls through a handle. Without leaving V86 mode code cannot be executed from EMBs; they serve only for data storage. The specification offers up to 64MB of extended memory divided into as many as 255 blocks.
- Upper Memory Blocks (UMBs). Block of memory located between 640KB and 1MB. Once a UMB is allocated, its memory is always available. Since the memory lies in conventional memory, code can be executed in it at any time.

The virtual Extended Memory Specification (XMS) device driver:

- Implements all the functions in XMS 2.0.
- Provides each DOS Session with a separate XMS emulation. Each DOS Session has its own high memory area, upper memory blocks, and extended memory blocks so that features such as interprocess communication perform as if each DOS Session was running on a different real 80386. A DOS Session cannot effect the availability of objects, or access memory in another DOS Session.
- Provides configurable limits for the amount of XMM memory available across DOS Sessions, and a limit per DOS Session. An installed program in the start list should be able to override the per-DOS Session limit, subject to the constraint given by the overall limit. In addition, it should be able to disable XMS completely for a particular DOS Session, if its installation would conflict with the program running in the DOS Session.

A CONFIG.SYS line of the form, `device=path\VXMS.SYS [options]`, will interpret *options* as "*/keyword=value*". These options are as follows:

/XMMLIMIT=g,i Set the global (system-wide) maximum memory usage of the virtual XMS device driver to *g*KB, and set a per-DOS Session maximum of *i*KB. These values are large enough to accommodate an automatic 64KB allocation in each DOS Session for the HMA. Values are restricted to the range, 0−65535 (=64KB).

The values of *g* and *i* are rounded up to the nearest multiple of 4. Specifying i=0 suppresses XMS installation in all DOS Sessions unless overridden by a DOS Session-specific configuration string. The default for /XMMLIMIT is *16384,2048*.

/HMAMIN=d Set the minimum request size (in kilobytes) for an HMA request to succeed. Values are restricted to the range, 0−63. The default for /HMAMIN is *0*.

/NUMHANDLES=n Set the number of handles available in each DOS Session. Each handle occupies 8 bytes. Values are restricted to the range, 0−128. The default for /NUMHANDLES is *32*.

/UMB Instruct XMM to create Upper Memory Blocks (UMB). The default is *off*.

/NOUMB Instruct XMM to not create Upper Memory Blocks (UMB). The default is */NOUMB*.

Virtual Mouse Device Driver

The virtual Mouse device driver (VMOUSE) is responsible for all mouse support in the Multiple DOS Sessions environment. Some of the most common services are listed below, and are available through the INT 33H interface:

- Driver state save, restore, and disable
- Light pen emulation
- Miscellaneous features
- Position and button event notification
- Position and button tracking
- Selectable pel and mickey mappings
- Video mode tracking
- Video pointer management (location and appearance).

There are no restrictions on any use of the INT 33H interface, even when a DOS Session is in background mode. The only difference between background, focused, and foreground DOS Session support is that no mouse events are posted for a background DOS Session. Mouse events are posted to full-screen DOS Session from the physical Mouse device driver, and are posted through the Presentation Manager* shield to focused windowed DOS Sessions. The pointer position should remain unchanged as long as a DOS Session is in background mode, unless an application initiates pointer movement of its own. In that case, the virtual pointer position changes but nothing is drawn (not even in virtual video space).

The virtual Mouse device driver also supports applications that use the BIOS INT 15H (AH=C2H) Pointing Device services.

INT 33H Support

INT 33H support is provided directly by the virtual Mouse device driver. If a DOS Session is running in a window (instead of full screen), and is in focus, the physical Mouse device driver routes the mouse events to the Presentation Manager single queue device driver, then to the shield, and eventually to the virtual Mouse device driver. The virtual Mouse device driver then buffers the mouse event as if it were for the full-screen DOS Session. When the DOS Setting, MOUSE_EXCLUSIVE_ACCESS, is *on* for a *windowed* DOS Session, all mouse events go directly from the physical Mouse device driver to the virtual Mouse device driver, bypassing the Presentation Manager interface. This property is necessary for those applications that track and draw their own mouse pointer.

Driver State Save, Restore, and Disable: Services are provided to copy all per-DOS Session mouse state information to an application-defined block and back again. When the driver is disabled, it ignores all INT 33H requests and physical Mouse device driver events.

Light Pen Emulation: Whenever this INT 33H feature is enabled, a virtual Mouse device driver INT 10H hook is established, and INT 10H Light Pen functions are emulated with mouse position and button data.

Miscellaneous Features: The INT 33H interface provides a function to read-and-reset *mouse motion counters*, which are simply software counters of total horizontal and vertical mickeys. These counters are simulated within the virtual Mouse device driver by taking reported pel movements and converting them to mickeys (based on the current mickey-to-pel ratio). This interface must also provide counts of button presses and releases, and the positions at the time of the last press and release. These are maintained by the virtual Mouse device driver without any assistance from the physical Mouse device driver.

* Trademark of the IBM Corporation

INT 15H (AH = C2H) Support

All of the published Pointing Device BIOS services (INT 15H, AH = C2H), are emulated in the virtual Mouse device driver. For a complete listing of these services, refer to the *Personal System/2 and Personal Computer BIOS Interface Technical Reference*.

Virtual Touch Device Driver

The virtual Touch device driver (VTOUCH) provides support for the INT 7FH for Multiple DOS Sessions. This virtual device driver is limited (by default) to making actual touch XYZ data available only to full screen DOS programs because the physical device driver, which handles the touch data interrupts, cannot determine which window to send the touch data to when running with the Presentation Manager session in the foreground.

In contrast, the physical Mouse device driver can determine which window to send the mouse data to because it is able to feed the single queue of the Presentation Manager. The Presentation Manager can then determine, based upon the event XYZ data, which window is to receive the event. If the window is a DOS window, it calls the virtual device driver.

For those DOS applications that receive full screen XYZ touch data while running in a window, the DOS Setting, TOUCH_EXCLUSIVE_ACCESS, is provided. This DOS Setting effectively turns off mouse emulation and directs all touch data to that window. The (now device-independent) INT 7FH continues to function in a window. However, touch data is not available to the application until it is switched to full screen. In all other respects, the application functions exactly as before.

DOS mouse applications are not subject to this restriction because they can run in a windowed or full screen DOS Session with touch input in the form of emulated mouse data.

INT 7FH Support

To access the touch display device, DOS applications use the software interrupt 7FH (INT 7FH) interface. OS/2 applications use the Tou*xxx* API. The virtual Touch device driver provides this by using a standard DOS Session software interrupt hook. This virtual device driver runs in protect mode (which allows it to share code in all DOS Sessions) and virtualizes all the hardware-dependent features that the INT 7FH interface provides (for example, changing thresholds). In addition to the touch device capability, it is also possible to use the touch display to emulate the IBM* Mouse. In this case, the standard mouse INT 33H API (for DOS) is used.

Notice that only DOS sessions running full screen can obtain full touch XYZ data. Windowed DOS sessions running under Presentation Manager are restricted to emulated mouse data. A DOS touch application that started as full screen stops receiving touch data if it is switched to a window, and starts receiving data again if it switched back to full screen. Conversely, a DOS touch application that is started in a window does not receive data until it is switched to full screen, and does not stop receiving data until it is switched back to a window.

* Trademark of the IBM Corporation

Virtual Video Device Drivers

The following list contains the installable virtual Video device drivers that are included with the Toolkit:

- Virtual CGA device driver (VCGA)
- Virtual EGA device driver (VEGA)
- Virtual MONO device driver (VMONO)
- Virtual VGA device driver (VVGA)
- Virtual XGA device driver (VXGA)
- Virtual 8514/A device driver (V8514A).

The architecture of the virtual Video device driver adheres to the architecture defined for all virtual device drivers. However, some portions of the Multiple DOS Sessions architecture have been designed primarily for (or used by) virtual Video device drivers:

- Foreground and background notification hooks
- Freeze and thaw services
- Title change and code-page change notification hooks.

A significant portion of the primary display virtual device drivers involves communication with the DOS Session Window Manager. This includes providing a set of services to fully describe a DOS Session's video state and a set of notifications to signal selected changes in state, so that a window remains relatively synchronized with its associated DOS Session.

Overview

Because the DOS Session Window Manager is a Ring 3 component, virtual device drivers run at Ring 0, and DOS Sessions have no LDT (and therefore cannot attach to DLLs), a Remote Procedure Call (RPC) mechanism must be used to communicate between the DOS Session Window Manager and a virtual device driver. The DOS Session Window Manager creates a thread for this purpose, which calls the virtual Video device driver to wait for a video event from any of the currently windowed DOS Sessions.

Virtual Video device drivers are base device drivers. DEVICE= in CONFIG.SYS lines are used to specify one or more virtual Video device drivers to load. The DOS Session Manager and DOS Session Window Manager identify a virtual Video device driver by its registered name (for example, a virtual Video device driver for the primary display device would register as VVIDEO1$, one for the secondary display device would register as VVIDEO2$, and so forth). After virtual Video device driver loading and initialization is complete, if no virtual device driver has registered for the primary display device, Multiple DOS Sessions support is disabled.

IBM-compatible CGA, EGA, VGA, XGA, and 8514/A adapters are supported as primary display devices, and MONO adapters are supported as secondary display devices. For these devices, all the standard configurations (that is, amount of video memory, type of monitor, and so forth), are supported. However, there are a a few exceptions. For example, support for other dual-adapter combinations, such as EGA/Mono with CGA/Color, or VGA/Mono with CGA/Color, are not specifically supported. In addition, all EGA monitor combinations (RGB, ECD, and MONO), EGA memory combinations (64KB, 128KB, 192KB, and 256KB), VGA and 8514/A monitor combinations (8512, 8513, and 8514), and VGA and XGA monitor combinations (8512, 8513, 8514, and 8515) are supported. The type of monitor used with a MONO or CGA adapter is immaterial, that is, it does not affect behavior of the adapter.

Virtual and Non-Virtual Operation

For optimal performance and compatibility, the virtual Video device drivers support full-screen operation. In this mode, there is no visual difference between DOS and DOS Session operation. For convenience, an option appears on the DOS Session system menu to convert the DOS Session from windowed mode to full-screen mode and back. Virtual Video device drivers support full-screen operation by performing the following operations:

- Registering foreground and background VIDEO_SWITCH_NOTIFICATION hooks with the DOS Session Manager
- Allocating a save and restore video buffer
- Installing I/O hooks to shadow key video port accesses
- Mapping physical video memory into the appropriate video address space
- Coercing text-mode fonts to match the currently selected code page
- Providing pointer drawing services to the virtual Mouse device driver to define, draw, and erase pointer images.

When a DOS Session does not own the physical display (that is, when it is background), the virtual Video device drivers:

- Install I/O hooks to track and emulate all video port accesses
- Map appropriate system memory to the active portions of the video address space
- Report video events to the DOS Session Window Manager (assuming the Presentation Manager is active and the DOS Session is an unminimized window). This includes changes to:
  - Mode
  - Palette
  - Cursor
  - Video memory
  - Input events
  - Scroll or string events
  - Paste continue or terminate (through the virtual Keyboard device driver)
  - Session title change
  - Screen-switch or video memory allocation errors.

At any time, the DOS Session Window Manager (or other processes that have access to a DOS Session's Process ID) can update a DOS Session's window state:

- Windowed or non-windowed
- Focus or non-focus
- Minimized or unminimized
- Lock or unlocked.

as well as query any aspect of a DOS Session's video state:

- Mode
- Palette
- Cursor
- Video memory
- Wait for video events
- Cancel wait for video events.

Mouse-Independent Pointer Drawing Services

When the virtual Video device driver creates a DOS Session for the first time, it opens the virtual Mouse device driver. If the open is successful, it provides the virtual Mouse device driver with the following entry points:

- Show Pointer
- Hide Pointer
- Define Text Pointer
- Define Graphics Pointer
- Set Video Page
- Set Light-Pen Emulation.

Also, whenever the DOS Session changes video modes (including the initial mode change that takes place during DOS Session creation), the virtual Video device driver notifies the virtual Mouse device driver of the new mode, the screen dimensions, and so forth.

Built-in Extended INT 10H and INT 2FH Services

The EGA Register interface is integrated into the virtual Video device driver as part of its INT 10H interception logic. This interface is a set of INT 10H services that are used by applications to make drawing operations and simultaneous use of the mouse pointer possible on EGA and VGA hardware. The virtual Video device driver's pointer drawing services are intended to replace this interface, but existing graphical applications still use it (generally when mouse support is also present), so the interface must be present or these applications will not function correctly.

The INT 2FH services notify applications when they are about to be switched to full-screen or background mode. Applications can use this notification to stop accessing video memory, if they are using a video memory not supported for background operation. This will prevent them from freezing, and also redraw their screen in the event the virtual Video device driver fails to fully restore it.

Summary of the 8514/A and XGA Virtual Device Drivers

DOS applications currently written to the 8514/A Adapter Interface (AI), the XGA AI, or directly to the 8514/A and XGA display adapters are able to run in a full-screen DOS Session. This is due to the ability of each adapter's virtual device driver to:

- Grant I/O privilege
- Save or restore the video state
- Provide video bitmap images to the Shield layer when the DOS Session is being windowed.

A DOS application is allowed to run only in a full-screen foreground DOS Session, and is immediately frozen if it attempts to access the hardware when switched to the background.

The virtual 8514/A device driver statements, which are required in CONFIG.SYS, are added during installation when the presence of an 8514/A display adapter is noted:

```
DEVICE=C:\OS2\MDOS\VVGA.SYS
DEVICE=C:\OS2\MDOS\V8514A.SYS
```

The virtual XGA device driver statements, which are required in CONFIG.SYS, are added during installation when the presence of an XGA display adapter is noted:

```
DEVICE=C:\OS2\MDOS\VVGA.SYS
DEVICE=C:\OS2\MDOS\VXGA.SYS
```

Listed below are the DOS Setting features of a DOS Session:

- Enable I/O trapping to allow the virtual device driver to save and restore the hardware instance used by a DOS application when the application switches away from the foreground DOS Session. The virtual device driver allocates a buffer to save the video image. The maximum size of this buffer is 1MB, and is obtained each time a full-screen DOS Session is created.

  Notice that enabling I/O trapping is the default. Although multiple instances of XGA can be present on a system, the virtual XGA device driver only saves or restores the state of the first instance used by a DOS application. It is the application's responsibility to save or restore the other XGA instances.

- Register for screen switch notification. This implies that a DOS application has hooked INT 2FH so as to be notified when it is to be switched to full-screen foreground or background. This notification informs the application it should restore the screen when switched to full-screen foreground, and should stop accessing video memory (to avoid freezing) when switched to background. An application that uses more than one instance of XGA should specify this option to save and restore the states of the other XGA adapters. The default is no VIDEO_SWITCH_NOTIFICATION.

Any DOS Session that is not saved or restored by the virtual device driver cannot be displayed in a window. DOS 8514/A and XGA applications are not interactive in a window; they can only be viewed.

Part 3. Reference Material

Chapter 5. Virtual DevHlp Services

Virtual DevHlp functions are used by virtual device drivers to access various services provided by the OS/2 operating system and by other virtual device drivers. These functions are provided because normal OS/2 API calls from a virtual device driver, and routines in a C run-time library, cannot be used for access purposes.

The Virtual DevHlp services are categorized below. Individual functions follow and are listed alphabetically with an explanation of their purpose, parameters, and calling conventions.

Some Notes on Virtual DevHlp Services

Passing Addresses of Buffers: The SSToDS macro must be used when passing the address of a frame (automatic) variable in a function call because the 32-bit code assumes SS = DS. Pointers allocated from the stack must be DS relative.

Function Definitions and Calling Conventions: All Virtual DevHlp services are 32-bit NEAR functions. For a description of the calling conventions, refer to the function prototype definition VDHENTRY located in the MVDM.INC include file included with the Toolkit.

When using the Virtual DevHlp services, the function being called (the callee) is responsible for cleaning up the stack. This means that when you call a Virtual DevHlp service, the function you call cleans up the stack; you do not have to pop the parameters off the stack after the call. This also means that when you register a hook (or callback) routine with a Virtual DevHlp service, your hook routine is responsible for cleaning up the stack before it returns to the function that called it.

The Virtual DevHlp services are called directly, as in `CALL VDHFunction`. This is in contrast to the physical device driver Device Helper services, which are called indirectly, as in `CALL [DevHlpFunction]`.

Syntax Examples: The syntax examples are shown in assembler language.

Defined Constants and Type Definitions: Defined constants, that is, *equate* statements in an include file, and type definitions are widely used in OS/2 2.0. These constant values are defined in the master include file, MVDM.INC, or in one of the files that MVDM.INC includes. The defined constants and type definitions that are seen in the syntax examples should be used to provide a degree of machine and version independence.

Invalid Parameters and System Halts: In many of the Virtual DevHlp functions, invalid parameters or other error conditions often *cause a system halt*. This is because virtual device drivers run at Ring 0 and have free access to everything in the system.

Remarks Section: Each function description has a **Remarks** section at the bottom, which consists of three parts:

- Context Issues: Indicates the specific contexts in which Virtual DevHlps should be called.
- DOS Session Termination Implications: Indicates what the virtual device driver must do upon termination of a DOS Session, for example, release resources.
- Notes: These are miscellaneous function-specific notes about the function.

Virtual DevHlp Services by Category

Virtual DevHlp services can be divided into categories based on the type of service the virtual DevHlp provides. These categories are:

DMA Services

- VDHAllocDMABuffer – Allocate DMA buffer (see page 5-7)
- VDHCallOutDMA – Call virtual DMA device driver (see page 5-22)
- VDHFreeDMABuffer – Free DMA buffer (see page 5-48)
- VDHRegisterDMAChannel – Register DMA channel (see page 5-121).

DOS Session Control Services

- VDHFreezeVDM – Freeze DOS Session (see page 5-52).
- VDHHaltSystem – Cause system halt (see page 5-59).
- VDHIsVDMFrozen – Determine if DOS Session is frozen (see page 5-74).
- VDHKillVDM – Terminate DOS Session (see page 5-75).
- VDHSetPriority – Adjust DOS Session scheduler priority (see page 5-146)
- VDHThawVDM – Allow frozen DOS Session to resume executing (see page 5-153)
- VDHYield – Yield the processor (see page 5-164).

DOS Settings Services

- VDHDecodeProperty – Decode property string (see page 5-34)
- VDHQueryProperty – Query virtual device driver property value (see page 5-109)
- VDHRegisterProperty – Register virtual device driver property (see page 5-123).

DPMI Services

- VDHArmVPMBPHook – Obtain address of DOS Session protect-mode breakpoint (see page 5-20)
- VDHBeginUseVPMStack – Begin using DOS Session protect-mode stack (see page 5-21)
- VDHChangeVPMIF – Change virtual Interrupt Flag (IF) (see page 5-23)
- VDHCheckPagePerm – Check Ring 3 page permissions (see page 5-24)
- VDHCheckVPMIntVector – Determine if DOS Session protect-mode handler exists (see page 5-25)
- VDHEndUseVPMStack – End use of DOS Session protect-mode stack (see page 5-43)
- VDHGetSelBase – Get flat base address (see page 5-56)
- VDHGetVPMExcept – Get DOS Session protect-mode exception vector (see page 5-57)
- VDHGetVPMIntVector – Return DOS Session protect-mode interrupt vector (see page 5-58)
- VDHRaiseException – Raise an exception to a DOS Session (see page 5-116)
- VDHReadUBuf – Read from protect-mode address space (see page 5-118)
- VDHSetVPMExcept – Set DOS Session protect-mode exception vector (see page 5-149)
- VDHSetVPMIntVector – Set DOS Session protect-mode interrupt vector (see page 5-150)
- VDHSwitchToVPM – Switch DOS Session to protect mode (see page 5-151)
- VDHSwitchToV86 – Switch DOS Session to V86 mode (see page 5-152)
- VDHWriteUBuf – Write to protect-mode address space (see page 5-162).

Note: "VPM," as found in the function names above, stands for *Virtual Protect Mode*.

File or Device I/O Services

- VDHClose – Close a file handle (see page 5-27)
- VDHDevIOCtl – Issue device-specific commands (see page 5-40)
- VDHOpen – Open a file or device (see page 5-81)
- VDHPhysicalDisk – Get information about partitionable disks (see page 5-88)

- VDHRead – Read bytes from file or device (see page 5-117)
- VDHSeek – Move read or write file pointer for a handle (see page 5-138)
- VDHWrite – Write bytes to file or device (see page 5-161).

GDT Selector Services

- VDHCreateSel – Create GDT selector to map a linear range (see page 5-32)
- VDHDestroySel – Destroy GDT selector (see page 5-37)
- VDHQuerySel – Get selector for data or stack address (see page 5-111).

Hook Management Services

- VDHAllocHook – Allocate hooks needed for interrupt simulation (see page 5-9)
- VDHArmBPHook – Obtain address of V86 breakpoint (see page 5-13)
- VDHArmContextHook – Set local or global context hook (see page 5-14)
- VDHArmReturnHook – Set handler to receive control (see page 5-15)
- VDHArmSTIHook – Set handler to receive control (see page 5-17)
- VDHArmTimerHook – Set a timer handler (see page 5-18)
- VDHFreeHook – Disarm and free hook (see page 5-49)
- VDHInstallIntHook – Set handler for V86 interrupt (see page 5-64)
- VDHInstallIOHook – Install I/O port hooks (see page 5-66)
- VDHInstallUserHook – Install handler for DOS Session event (see page 5-70)
- VDHQueryHookData – Returns pointer to hook reference data (see page 5-106)
- VDHRemoveIOHook – Remove hooks for PIC I/O ports (see page 5-132)
- VDHSetIOHookState – Enable/disable I/O port trapping (see page 5-144).

Idle DOS Application Management Services

- VDHReportPeek – Report DOS Session polling activity (see page 5-133)
- VDHWakeIdle – Wake up DOS Session (see page 5-159).

Note: The services above indicate when a DOS application appears to be idle, and when activity exists that could make the DOS application *busy*.

Inter-Device Communication Services

- VDHCloseVDD – Close virtual device driver (see page 5-28)
- VDHOpenPDD – Open physical device driver (see page 5-83)
- VDHOpenVDD – Open virtual device driver (see page 5-85)
- VDHRegisterVDD – Register virtual device driver entry points (see page 5-127)
- VDHRequestVDD – Issue request for virtual device driver operation (see page 5-135).

Keyboard Services

- VDHQueryKeyShift – Query keyboard shift state (see page 5-108).

Memory Management Services: There are three sub-categories of the memory management Virtual DevHlp services. The first two are used for the granularity of the memory allocation unit, the third category is used for memory locking services.

Byte Granular Memory Management Services

- VDHAllocBlock – Allocate block from memory block pool (see page 5-6)
- VDHAllocDOSMem – Allocate block of memory from DOS area (see page 5-8)
- VDHAllocMem – Allocate small amount of memory (see page 5-10)
- VDHCopyMem – Copy from one linear memory address to another (see page 5-30)
- VDHCreateBlockPool – Create memory block pool (see page 5-31)
- VDHDestroyBlockPool – Destroy memory block pool (see page 5-36)

- VDHExchangeMem – Exchange contents of two linear memory regions (see page 5-45)
- VDHFreeBlock – Free block of memory (see page 5-47)
- VDHFreeMem – Free memory (see page 5-50).

Page Granular Memory Management Services

- VDHAllocPages – Allocate page-aligned memory object (see page 5-11)
- VDHFindFreePages – Find largest available linear memory region (see page 5-46)
- VDHFreePages – Free memory object (see page 5-51)
- VDHGetDirtyPageInfo – Return status of dirty bits (see page 5-54)
- VDHInstallFaultHook – Install page fault handler (see page 5-62)
- VDHMapPages – Map specified linear address (see page 5-78)
- VDHQueryFreePages – Return amount of free virtual memory (see page 5-105)
- VDHReallocPages – Reallocate memory object (see page 5-120)
- VDHRemoveFaultHook – Remove page fault handler (see page 5-131)
- VDHReservePages – Reserve range of linear addresses (see page 5-136)
- VDHUnreservePages – Unreserve range of linear addresses (see page 5-155).

Memory Locking Memory Management Services

- VDHLockMem – Verify access or lock memory (see page 5-76)
- VDHUnlockMem – Release memory lock (see page 5-154).

Note: The services above allow virtual device drivers to allocate, free, reallocate, and lock memory for global, per-DOS Session, page granular, or byte granular objects. Four types of mappings are supported:

- Mapping to a physical address
- Mapping to another linear address
- Mapping to black holes (don't care) pages
- Mapping to invalid (unmapped) pages.

Virtual device drivers can also request smaller memory allocations from the kernel heap, which is global and fixed. Small, fixed size block services are available to speed up frequent allocations and freeing of memory. For a particular block size, a pool of blocks is maintained and the requirements are met by taking off a block from the block pool.

Miscellaneous Virtual DevHlp Services

- VDHDevBeep – Device beep Virtual DevHlp service (see page 5-39)
- VDHEnumerateVDMs – Run worker function for each DOS Session (see page 5-44)
- VDHGetCodePageFont – Return information on DOS Session code page font (see page 5-53)
- VDHGetError – Get error code from last Virtual DevHlp service (see page 5-55)
- VDHHandleFromPID – Get handle for given Process ID (see page 5-60)
- VDHHandleFromSGID – Get DOS Session handle from Screen Group ID (see page 5-61)
- VDHPopup – Display message (see page 5-94)
- VDHPutSysValue – Set system value (see page 5-103)
- VDHQueryA20 – Query current state of A20 (see page 5-104)
- VDHQueryLin – Get linear address for Far16 address (see page 5-107)
- VDHQuerySysValue – Query system value (see page 5-113)
- VDHReleaseCodePageFont – Release code page font (see page 5-129)
- VDHSetA20 – Enable or disable A20 for current DOS Session (see page 5-140)
- VDHSetDosDevice – Register/install DOS device driver (see page 5-141)
- VDHSetError – Set error code (see page 5-142)
- VDHSetFlags – Set DOS Session Flags register (see page 5-143).

Parallel Port and Printer Services

- VDHPrintClose – Flush and close open printers for DOS Session (see page 5-97).

Semaphore Services

- VDHCreateSem – Create event or mutex semaphore (see page 5-33)
- VDHDestroySem – Destroy semaphore (see page 5-38)
- VDHPostEventSem – Post event semaphore (see page 5-96)
- VDHQuerySem – Query semaphore state (see page 5-112)
- VDHReleaseMutexSem – Release mutex semaphore (see page 5-130)
- VDHRequestMutexSem – Request mutex semaphore (see page 5-134)
- VDHResetEventSem – Reset event semaphore (see page 5-137)
- VDHWaitEventSem – Wait on event semaphore (see page 5-156).

Note: The services above are used for synchronizing with an OS/2 process. Virtual device drivers must be careful not to block (VDHRequestMutexSem/VDHWaitEventSem) in the context of a DOS Session task, otherwise, that task will receive no more simulated hardware interrupts until it becomes unblocked.

Timer Services

- VDHArmTimerHook – Set timer service/handler (see page 5-18)
- VDHDisarmTimerHook – Cancel timer service (see page 5-42).

Virtual Interrupt Services

- VDHClearVIRR – Clear virtual IRR (see page 5-26)
- VDHCloseVIRQ – Deregister IRQ handler (see page 5-29)
- VDHOpenVIRQ – Register IRQ handler (see page 5-86)
- VDHQueryVIRQ – Query IRQ status in a DOS Session (see page 5-115)
- VDHSendVEOI – Send virtual EOI to VPIC (see page 5-139)
- VDHSetVIRR – Set virtual Interrupt Request Register (IRR) (see page 5-148)
- VDHWaitVIRRs – Wait until interrupt is simulated (see page 5-157)
- VDHWakeVIRRs – Wake up DOS Session (see page 5-160).

V8086 Stack Manipulation Services

- VDHPopInt – Remove IRET Frame from Client DOS Session Stack (see page 5-90)
- VDHPopRegs – Pop Client DOS Session Registers from Client Stack (see page 5-91)
- VDHPopStack – Pop Data off Client Stack (see page 5-93)
- VDHPushFarCall – Simulate FAR Call to V86 Code (see page 5-98)
- VDHPushInt – Transfer Control to V86 Interrupt Handler (see page 5-99)
- VDHPushRegs – Push Client DOS Session Registers onto Client Stack (see page 5-100)
- VDHPushStack – Push Data onto Client Stack (see page 5-102).

VDHAllocBlock

This function allocates a block from the specified memory block pool. This block is allocated from fixed or swappable memory, depending on the value of the OptionFlag flag when the block is created with VDHCreateBlockPool.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHAllocBlock:NEAR
SourceBlockPool   DD     ?    ; Handle to a memory block pool

PUSH   SourceBlockPool        ; Push the parameter

CALL   VDHAllocBlock          ; Call the function
                              ; Notice that it isn't necessary to pop the variable that was
                              ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
SourceBlockPool	DD	Handle to the pool of memory blocks from which to allocate a memory block.

Results

Success If the function is successful, it returns the address of the allocated memory block.

Failure If the function fails, it returns 0. VDHGetError should be called to determine the nature of the problem. Notice that passing an invalid memory block handle *causes a system halt.*

Remarks:

Context Issues: This function can be called in the initialization or task context.

DOS Session Terminations: Any blocks allocated by a virtual device driver for use by the terminating DOS Session are freed by using VDHFreeBlock. Any block pools created for the DOS Session should also be freed.

VDHAllocDMABuffer

This function allocates a DMA buffer.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHAllocDMABuffer:NEAR
RequestSize      DD     ?      ; Request size
Align64kFlag     DD     ?      ; Alignment flag
PhysAddrPtr      DD     ?      ; Location to store the physical address returned

PUSH    RequestSize            ; Push the parameters
PUSH    Align64kFlag           ;
PUSH    PhysAddrPtr            ;

CALL   VDHAllocDMABuffer       ; Call the function
                               ; Notice that it isn't necessary to pop the variable that was
                               ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
RequestSize	DD	Request size. See below.
Align64kFlag	DD	Alignment flag. See below.
PhysAddrPtr	DD	Location where VDHAllocDMABuffer returns the physical address of the allocation.

RequestSize If the Align64kFlag=*1*, RequestSize is always less than, or equal to, 64KB. If the Align64kFlag=*0*, RequestSize is always be less than, or equal to, 128KB.

Align64kFlag If Align64kFlag=*1*, the buffer is at a 64KB alignment in physical memory. If Align64kFlag=*0*, the buffer is at a 128KB alignment in physical memory.

Results

Success If the function is successful, it returns the linear address of allocation (system memory).

Failure If the function fails, it returns *–1*. VDHGetError should be called to determine the nature for the problem.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: The virtual device driver frees the buffer by using VDHFreeDMABuffer at DOS Session termination.

Notes: Physical memory is allocated resident and contiguous, and is always in the *first* 16MB. The Linear Memory range comes from the system arena. The contents of the returned buffer are not important, that is, the buffer is not zero filled.

VDHAllocDOSMem

This function allocates a block of memory from the DOS memory area. Allocations start at *0* and go to *256KB*.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHAllocDOSMem:NEAR
BlockSize     DD     ?     ; Size of the desired memory block (in bytes)

PUSH   BlockSize           ; Push the parameter

CALL   VDHAllocDOSMem      ; Call the function
                           ; Notice that it isn't necessary to pop the variable that was
                           ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
BlockSize	DD	Size of the desired memory block; the number of bytes to allocate.

Results

Success If the function is successful, it returns the address of the allocated memory block.

Failure If the function fails, it returns 0. VDHGetError should be called to determine the nature of the problem. If BlockSize = *0*, or if the function is called in an invalid context (see *Context Issues* below), *a system halt occurs*.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context (DOS Session creation).

DOS Session Terminations: There are no DOS Session termination implications for this function. The DOS Session Manager frees this memory.

Notes: There is no way to free memory taken from the DOS heap. If there is no free memory in the DOS memory area, the allocation fails. Allocations always start on paragraph (16-byte) boundaries. Allocations never actually start at *0* because system virtual device drivers allocate DOS memory during DOS Session initialization for an Interrupt Vector table, a BIOS data area, and a DOS communication area.

VDHAllocHook

This function is used to allocate a hook handle for the arm hook services, Context, STI, Return, Timer, and BP (breakpoint).

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHAllocHook:NEAR
HookType        DD      ?       ; Hook type
ArmHookFcn      DD      ?       ; Arm hook function
RefDataSize     DD      ?       ; Size of the reference data

PUSH   HookType                 ; Push the parameters
PUSH   ArmHookFcn               ;
PUSH   RefDataSize              ;

CALL   VDHAllocHook             ; Call the function
                                ; Notice that it isn't necessary to pop the variable that was
                                ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
HookType	DD	Hook type. See below.
ArmHookFcn	DD	Arm hook function.
RefDataSize	DD	Size of the reference data.

HookType Hook type. Possible values are:

VDH_CONTEXT_HOOK	VDHArmContextHook
VDH_STI_HOOK	VDHArmSTIHook
VDH_RETURN_HOOK	VDHArmReturnHook
VDH_TIMER_HOOK	VDHArmTimerHook
VDH_BP_HOOK	VDHArmBPHook.

Results

Success If the function is successful, it returns a hook handle, which is used in the Arm services.

Failure If there is an error, the function returns NULL. VDHGetError should be called to determine the nature of the problem. If HookType or ArmHookFcn is invalid, *a system halt occurs*.

Remarks:

Context Issues: This function can be called in the initialization or DOS Session-task context. However, VDH_RETURN_HOOK requires the DOS Session-task context.

DOS Session Terminations: During initialization time, hooks are allocated globally; that is, they are not associated with a DOS Session process. These hooks are not automatically freed when a DOS Session terminates. During DOS Session creation time or thereafter, allocated hooks are associated with the current DOS Session, and are freed when that DOS Session terminates.

VDHAllocMem

This function allocates a small amount of memory on a LONG/DWORD boundary. The memory is allocated from the system area so the address is valid in all process contexts.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHAllocMem:NEAR
NumBytes       DD     ?       ; Number of bytes to allocate
OptionFlag     DD     ?       ; Option bit flag

PUSH   NumBytes               ; Push the parameters
PUSH   OptionFlag             ;

CALL   VDHAllocMem            ; Call the function
                              ; Notice that it isn't necessary to pop the variable that was
                              ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
NumBytes	DD	Number of bytes to allocate.
OptionFlag	DD	Bit flag describing the allocation options. Possible value: **VDHAM_SWAPPABLE** Allocate memory from the swappable heap. Otherwise, memory is allocated from the fixed heap and can be accessed at hardware interrupt time.

Results

Success If the function is successful, it returns the address of the allocated space.

Failure If the function fails, it returns *0*. VDHGetError should be called to determine the nature of the problem. If either NumBytes or OptionFlag is invalid, *a system halt occurs.*

Remarks:

Context Issues: This function can be called in the initialization or task context.

DOS Session Terminations: All memory allocated by a virtual device driver for use by the terminating DOS Session is freed by using VDHFreeMem.

Notes: Allocations larger than one page are performed by using VDHAllocPages. To access the allocated memory at hardware interrupt time, be sure to allocate memory from the fixed heap.

VDHAllocPages

This function allocates a page-aligned, page-granular memory object.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHAllocPages:NEAR
StartingAddress        DD     ?     ; A specific linear address
NumPages               DD     ?     ; The number of pages to allocate
OptionFlag             DD     ?     ; Allocation options bit-flag

PUSH   StartingAddress              ; Push the parameters
PUSH   NumPages                     ;
PUSH   OptionFlag                   ;

CALL   VDHAllocPages                ; Call the function
                                    ; Notice that it isn't necessary to pop the variable that was
                                    ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
StartingAddress	DD	A specific address used with the available options (see OptionFlag).
NumPages	DD	Number of pages of linear memory to allocate.
OptionFlag	DD	Bit flag indicating choices in allocation options. See below.

OptionFlag This flag can take one or more of the following values:

VDHAP_SPECIFIC VDHAllocPages attempts to allocate pages starting at the linear address, StartingAddress. Otherwise, an arbitrary linear address is selected.

VDHAP_SYSTEM The linear address is allocated from the system (global) area, essentially allocating global memory. Otherwise, it is allocated from the private area of this process.

VDHAP_FIXED The allocated memory is fixed. Otherwise, the memory is swappable. This flag cannot be used if VDHAP_PHYSICAL is specified.

VDHAP_PHYSICAL The allocated linear address range maps the physical address range, starting at StartingAddress. Otherwise, an arbitrary physical pages is assigned. Notice that this flag cannot be used if VDHAP_FIXED is specified.

Results

Success If the function is successful, it returns the address of the allocated memory.

Failure If the function fails, it returns 0. VDHGetError should be called to determine the nature of the problem. If the OptionFlag flag is invalid, *a system halt occurs.*

Remarks:

Context Issues: This function can be called in the initialization or task context. If called in the initialization context, VDHAP_SYSTEM and VDHAP_PHYSICAL must be specified.

DOS Session Terminations: Any allocations that are not in the terminating DOS Sessions private area must be released by using VDHFreePages.

Notes: When an allocation made with VDHAllocPages is shrunk by VDHReallocPages, the linear range between the end of the allocation and the original end of the allocation remains available for object growth without movement. Regardless of VDHReallocPages activity, all pages in the allocation retain the same properties (that is, fixed, system, and physical).

VDHArmBPHook

This function obtains the address of the V86 breakpoint allocated by a previous call to VDHAllocHook with the VDH_BP_HOOK flag.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHArmBPHook:NEAR
HookHandle     DD     ?        ; Hook handle

PUSH   HookHandle              ; Push the parameter

CALL   VDHArmBPHook            ; Call the function
                               ; Notice that it isn't necessary to pop the variable that was
                               ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
HookHandle	DD	Hook handle allocate with VDHAllocHook.

Hook routine interface:

```
HookRoutine  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 8] - p    - Pointer to hook-specific reference data
;           [ESP + 4] - pcrf - Pointer to client register frame
;    EXIT   None
;
; CONTEXT   DOS Session-task
```

Results

Success If the function is successful, it returns a V86 breakpoint address.

Failure If HookHandle is invalid, *a system halt occurs*.

Remarks:

Context Issues: This function can be called only in the task context, and needs to be called only at DOS Session creation time.

DOS Session Terminations: There are no DOS Session termination implications for this function. BP (breakpoint) hooks are automatically removed during DOS Session termination.

Notes: This VDH service provides a raw breakpoint service. Upon entry to the arm hook function, the client's CS:IP is randomly set. Therefore, it is crucial that this function set the CS:IP to a valid address. Typically, the CS:IP is retrieved from the client's stack through use of VDHPopStack. This is a low-level service. Appropriate care should be exercised to ensure that the client's CS:IP and stack remain valid.

Unlike other hooks, breakpoint hooks are armed until explicitly disarmed by a call to VDHFreeHook. When the V86 breakpoint address is called from V86 mode, the hook function is called. Any changes made to the client register frame cause the corresponding registers to take on these values upon return to V86 mode. The *p* parameter in the hook routine points to the cbRefData bytes of data allocated by VDHAllocHook.

VDHArmContextHook

This function is used to get to a task-time context from interrupt time when an interrupt is simulated. This service adds a handler to the list of routines called the next time the task context is entered (global context), or the next time a specific DOS Session-task context is reached (local context). This is done only once; the handler is removed from the list after it is executed.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHArmContextHook:NEAR
HookHandle     DD     ?      ; Hook handle allocated with VDHAllocHook
VDMHandle      DD     ?

PUSH   HookHandle            ; Push the parameters
PUSH   VDMHandle             ;

CALL   VDHArmContextHook     ; Call the function
                             ; Notice that it isn't necessary to pop the variable that was
                             ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
HookHandle	DD	Hook handle allocated with VDHAllocHook.
VDMHandle	DD	DOS Session handle. A value of *−1* indicates a global context.

Hook routine interface:

```
HookRoutine  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 8] - pRefData - Pointer to reference data
;           [ESP + 4] - pcrf     - Pointer to client register frame
;    EXIT   None
;
; USES       EAX, ECX, EDX, FS, GS, Flags
; CONTEXT    Task (global) or DOS Session-task (local)
; NOTE       pcrf is valid only for local context hook handlers.
```

Results

Success If the function was successful, it returns nothing.

Failure If HookHandle or VDMHandle is invalid or if the hook is already armed, *a system halt occurs.*

Remarks:

Context Issues: This function can be called in the task or interrupt context.

DOS Session Terminations: Any context hooks allocated in the context of the terminating DOS Session are deallocated by the DOS Session Manager.

Notes: After calling this routine, the caller is guaranteed that the context hook will be executed before any user-space code (global context) or any V86 code in the specified DOS Session (local context).

VDHArmReturnHook

This function sets a service to receive control when an Interrupt Return (IRET) or Far Return (RETF) instruction is executed in V86 mode or when VDHPopInt is executed.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHArmReturnHook:NEAR
HookHandle      DD     ?       ; Hook handle
RetHookType     DD     ?       ; Return hook type

PUSH   HookHandle              ; Push the parameters
PUSH   RetHookType             ;

CALL   VDHArmReturnHook        ; Call the function
                               ; Notice that it isn't necessary to pop the variable that was
                               ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
HookHandle	DD	Hook handle the routine to get control on an IRET/RETF. Allocated with VDHAllocHook.
RetHookType	DD	Return hook type. See below.

RetHookType Return hook type. Possible values are:

VDHARH_NORMAL_IRET
VDHARH_NORMAL_RET
VDHARH_RECURSIVE_IRET
VDHARH_RECURSIVE_RET
VDHARH_CSEIP_HOOK
VDHARH_RECURSIVE_CSEIP_HOOK.

Hook routine interface:

```
HookRoutine  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 8] - pRefData - Pointer to reference data
;           [ESP + 4] - pcrf     - Pointer to client register frame
;    EXIT   None
;
; USES      EAX, ECX, EDX, FS, GS, Flags
; CONTEXT   DOS Session-task.
```

Results

Success If the function was successful, it returns a non-zero value.

Failure If VDHARH_NORMAL_x is already armed, the function returns 0. If HookHandle is invalid, *a system halt occurs.*

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: Any return hooks allocated in the context of the terminating DOS Session are deallocated automatically by the DOS Session Manager.

Notes: The VDHARH_NORMAL_RET and VDHARH_RECURSIVE_RET return hook types can be made only after VDHPushFarCall has been called but before returning to V86 mode. The return hook types, VDHARH_NORMAL_IRET and VDHARH_RECURSIVE_IRET, can be made only in the following contexts:

- Within a software interrupt handler set by VDHInstallIntHook
- After VDHPushInt has been called but before returning to V86 mode.

The VDHARH_RECURSIVE_x return types push two extra WORDs on the client's stack before the IRET or RET frame. VDHARH_CSEIP_HOOK and VDHARH_RECURSIVE_CSEIP_HOOK are meant to be used before a VDHPushInt or VDHPushFarCall as it is more efficient to do this type of return hook first. Only the normal and recursive CS:EIP hooks will work in DOS Session protect mode.

VDHArmSTIHook

This function sets a handler to receive control when the current DOS Session enables simulated interrupts. This handler is called only once. If interrupts are already enabled in the DOS Session, the handler is called immediately.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHArmSTIHook:NEAR
HookHandle      DD      ?       ; Hook handle allocated with VDHAllocHook
VDMHandle       DD      ?       ; DOS Session handle

PUSH   HookHandle               ; Push the parameters
PUSH   VDMHandle                ;

CALL   VDHArmSTIHook            ; Call the function
                                ; Notice that it isn't necessary to pop the variable that was
                                ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
HookHandle	DD	Hook handle allocated with VDHAllocHook.
VDMHandle	DD	DOS Session handle.

Hook routine interface:

```
HookRoutine  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 8] - pRefData - Pointer to reference data
;           [ESP + 4] - pcrf     - Pointer to client register frame
;    EXIT   None
;
; USES      EAX, ECX, EDX, FS, GS, Flags
; CONTEXT   DOS Session-task.
```

Results

Success If the function was successful, it returns nothing.

Failure If either HookHandle or VDMHandle is invalid or if the hook is already armed, *a system halt occurs*.

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: Any STI hooks allocated in the context of the terminating DOS Session are deallocated automatically by the DOS Session Manager.

Notes: Interrupt re-enabling detection is performed when the DOS Session runs with IOPL=*0*. This causes any instructions that might modify the interrupt flag to fault.

VDHArmTimerHook

This function sets a timer handler to be called after a specified duration has elapsed. The handler is called only once.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHArmTimerHook:NEAR
HookHandle     DD     ?     ; Timer hook handle
Duration       DD     ?     ; Duration of the timeout in milliseconds
VDMHandle      DD     ?     ; DOS Session handle

PUSH   HookHandle           ; Push the parameters
PUSH   Duration             ;
PUSH   VDMHandle            ;

CALL   VDHArmTimerHook      ; Call the function
                            ; Notice that it isn't necessary to pop the variable that was
                            ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
HookHandle	DD	Timer hook handle.
Duration	DD	Timer duration in milliseconds.
VDMHandle	DD	DOS Session handle. See below.

VDMHandle Possible values are:

DOS Session Handle or 0	When the specified time duration expires, a global context hook is automatically armed by using the specified HookHandle.
VDH_TIMER_GLOBAL_CONTEXT	When the specified time duration expires, a local context hook is automatically armed by using the specified HookHandle.
VDH_TIMER_INTERRUPT_HOOK	When the specified time duration expires, the timer hook specified by HookHandle is called at interrupt time.

Hook routine interface:

```
HookRoutine  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 4] - pRefData - Pointer to reference data
;    EXIT   None
;
; CONTEXT   Interrupt.
```

Results

Success If the function is successful, it returns nothing.

Failure If either Duration or HookHandle is invalid or the hook is already armed, *a system halt occurs.*

Remarks:

Context Issues: This function can be called in the initialization, task, or interrupt context.

DOS Session Terminations: Any timer hooks allocated in the context of the terminating DOS Session are deallocated automatically by the DOS Session Manager at DOS Session termination.

Notes: If of type, VDH_TIMER_INTERRUPT_HOOK, the timer handler cannot block since it executes at physical interrupt time.

VDHArmVPMBPHook

This function obtains the address of the DOS Session's protect-mode breakpoint allocated by a previous call to VDHAllocHook with the VDH_BP_HOOK flag.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHArmVPMBPHook:NEAR
HookHandle     DD     ?

PUSH   HookHandle              ; Push the parameter

CALL   VDHArmVPMBPHook         ; Call the function
                               ; Notice that it isn't necessary to pop the variable that was
                               ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
HookHandle	DD	Hook handle allocated with VDHAllocHook.

The hook routine interface:

```
HookRoutine  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 8] - p    - Pointer to hook-specific reference data
;           [ESP + 4] - pcrf - Pointer to client register frame
;    EXIT   None
;
; CONTEXT   DOS Session-task.
```

Results

Success If the function is successful, it returns a DOS Session protect-mode breakpoint address.

Failure If HookHandle is invalid, *a system halt occurs.*

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: Unlike other hooks, breakpoint hooks are armed until explicitly disarmed by a call to VDHFreeHook. When the breakpoint address is executed from DOS Session protect mode, the hook function is called. Any changes made to the client register frame cause the corresponding registers to take on these values after returning to the DOS Session. The *p* parameter in the hook routine points to the RefDataSize bytes of data allocated by VDHAllocHook.

VDHArmVPMBPHook provides a raw breakpoint service. On entry to the arm hook function, the client's CS:EIP is randomly set. Therefore, it is crucial that the virtual device driver's hook handler set the CS:EIP to a valid address. Typically, the CS:EIP is retrieved from the client's stack through the use of VDHPopStack, which is a low-level service. Appropriate care should be exercised to ensure that the client's CS:EIP and stack remain valid.

VDHBeginUseVPMStack

This function is used to switch to the DOS Session's protect-mode stack for hardware interrupts, exceptions, and so forth. This service can be called repeatedly but only the first call will switch stacks. Subsequent calls increment use count but remain on the VPM stack.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHBeginUseVPMStack:NEAR

CALL   VDHBeginUseVPMStack     ; Call the function
                               ; Notice that it isn't necessary to pop the variable that was
                               ; pushed; the Virtual DevHlp function does this.
```

Parameters: None.

Results: None.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHCallOutDMA

If a virtual device driver owns a DMA channel, it must call the virtual DMA device driver for those interrupts that occur when a DMA request is pending on that channel. This function is used by the virtual DMA device driver (VDMA) to check how much DMA transfer has been completed, and to copy the contents from the temporary buffer to the DOS area. This service also indicates when the DOS Session is done with the channel so that the channel can be given to another waiting DOS Session.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHCallOutDMA:NEAR

CALL   VDHCallOutDMA          ; Call the function
                              ; Notice that it isn't necessary to pop the variable that was
                              ; pushed; the Virtual DevHlp function does this.
```

Parameters: None.

Results: None.

Remarks:

Context Issues: This function can be called in any context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHChangeVPMIF

This function changes the virtual Interrupt Flag (IF) that enables or disables protect-mode interrupts. This change is reflected in a bit in the flVDMStatus kernel variable (see "Notes" below). If the STI hooks are waiting and interrupts are enabled, the hooks are called.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHChangeVPMIF:NEAR
SetIFFlag    DD     ?

PUSH   SetIFFlag          ; Push the parameter

CALL   VDHChangeVPMIF     ; Call the function
                          ; Notice that it isn't necessary to pop the variable that was
                          ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
SetIFFlag	DD	Indicates whether to turn the IF flag *on* or *off*. If non-zero, set IF flag to *1*. If zero, set IF flag to *0*.

Results

Success If the function is successful, it returns a non-zero value.

Failure Not applicable to this function.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: The flVDMStatus kernel variable is exported from the kernel. Any virtual device driver can refer to it directly. flVDMStatus is a DD-sized variable that contains various DOS Session Status flags relating to protect mode and the DOS Protect Mode Interface (DPMI).

The flVDMStatus flags indicate:

- DPMI was initialized (protect mode was enabled)
- 16-bit or 32-bit
- The current execution mode (V86 or protect).

These flags and their values are:

```
VDM_STATUS_VPM_32          0x00000001     /* 32-bit DPMI application      */
VDM_STATUS_VPM_APP         0x00000002     /* DPMI application started     */
VDM_STATUS_VPM_EXEC        0x00000004     /* In DOS Session protect mode  */
VDM_STATUS_VPM_TRAPMODE    0x00000008     /* V86 mode at kernel entry?    */
VDM_STATUS_VPM_IF_FLAG     0x00000010     /* Virtual IF flag              */
VDM_STATUS_VPM_08_HOOK     0x00000020     /* Protect 08H hooked?          */
VDM_STATUS_VPM_1C_HOOK     0x00000040     /* Protect 1CH hooked?          */
VDM_STATUS_VPM_PERM        0x00000080     /* Protect mode allowed?        */
VDM_STATUS_VPM_XDOS        0x00000100     /* DOS API Extension active?    */
VDM_STATUS_VPM_TERM        0x00000200     /* DOS Session terminating?     */
```

VDHCheckPagePerm

This function checks Ring 3 page permissions for a range of pages. This service fails if a Ring 3 client faults on any of the pages that have been checked.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHCheckPagePerm:NEAR
BasePage     DD     ?
Reserved     DD     0       ; Reserved; must be set to zero.
NumPages     DD     ?
Flag         DD     ?

PUSH   BasePage             ; Push the parameters
PUSH   Reserved             ;
PUSH   NumPages             ;
PUSH   Flag                 ;

CALL   VDHCheckPagePerm     ; Call the function
                            ; Notice that it isn't necessary to pop the variable that was
                            ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
BasePage	DD	Base virtual page number.
Reserved	DD	Reserved. Must be set to *0*.
NumPages	DD	Number of pages to verify.
Flag	DD	Values are: VPMPG_U Are the pages user accessible? VPMPG_W Are the pages writable? VPMPG_R Are the pages readable (valid)? VPMPG_X Are the pages executable?

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, it returns 0.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHCheckVPMIntVector

This function returns a non-zero value if the application protect-mode interrupt vector is set.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHCheckVPMIntVector:NEAR
Vector     DD     ?

PUSH   Vector                 ; Push the parameter

CALL   VDHCheckVPMIntVector   ; Call the function
                              ; Notice that it isn't necessary to pop the variable that was
                              ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
Vector	DD	Interrupt vector number.

Results

Success If the function is successful, it returns a non-zero value.

Failure If there is no user-registered interrupt handler for the interrupt in Vector, the function returns 0.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHClearVIRR

This function clears the virtual Interrupt Request Register (IRR) in the Virtual Programmable Interrupt Controller (VPIC) of the specified DOS Session for the IRQ specified. The simulation of interrupts to the specified DOS Session is stopped on this IRQ level.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHClearVIRR:NEAR
VDMHandle     DD     ?     ; DOS Session handle
IRQHandle     DD     ?     ; IRQ handle from VDHOpenVIRQ

PUSH   VDMHandle           ; Push the parameters
PUSH   IRQHandle           ;

CALL   VDHClearVIRR        ; Call the function
                           ; Notice that it isn't necessary to pop the variable that was
                           ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
VDMHandle	DD	DOS Session handle. A value of *0* indicates the current DOS Session.
IRQHandle	DD	IRQ handle from VDHOpenVIRQ.

Results

Success If the function was successful, it returns nothing.

Failure If either of the parameters is invalid, *a system halt occurs.*

Remarks:

Context Issues: This function can be called in the task or interrupt context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHClose

This function closes a file or device that was opened with VDHOpen.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHClose:NEAR
FileHandle     DW     ?      ; Handle to the file/device to close

PUSH   FileHandle            ; Push the parameter

CALL   VDHClose              ; Call the function
                             ; Notice that it isn't necessary to pop the variable that was
                             ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
FileHandle	DW	Handle (from VDHOpen) to the file or device to close.

Results

Success If the function is successful, it returns nothing.

Failure If FileHandle is invalid or if the DOS Session is not in the DOS Session-task context, *a system halt occurs*.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: Any handles opened by the virtual device driver for the terminating DOS Session must be closed.

VDHCloseVDD

This function closes a virtual device driver/virtual device driver (VDD/VDD) Inter-Device Communication (IDC) session that was opened with VDHOpenVDD.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHCloseVDD:NEAR
VDDHandle     DD     ?     ; Handle of the virtual device driver to close

PUSH   VDDHandle           ; Push the parameter

CALL   VDHCloseVDD         ; Call the function
                           ; Notice that it isn't necessary to pop the variable that was
                           ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
VDDHandle	DD	Handle (from VDHOpenVDD) to the virtual device driver to close.

Results

Success If the function is successful, it returns nothing.

Failure If VDMHandle is invalid, *a system halt occurs.*

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: If a virtual device driver has a VDD/VDD session for a particular DOS Session, that session is closed when the DOS Session terminates.

Notes: This service is used to close the type of VDD/VDD connections, which are temporary or are on a per-DOS Session basis.

VDHCloseVIRQ

This function closes the virtual IRQ handle passed to it.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHCloseVIRQ:NEAR
IRQHandle     DD     ?       ; Handle from VDHOpenVIRQ

PUSH   IRQHandle             ; Push the parameter

CALL   VDHCloseVIRQ          ; Call the function
                             ; Notice that it isn't necessary to pop the variable that was
                             ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
IRQHandle	DD	Handle to the IRQ (from VDHOpenVIRQ) to close.

Results

Success If the function was successful, it returns nothing.

Failure If IRQHandle is invalid, *a system halt occurs*.

Remarks:

Context Issues: This function can be called only in the initialization context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHCopyMem

This function copies memory from one user linear address to another.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHCopyMem:NEAR
Source          DD     ?      ; Address of the data to copy
Destination     DD     ?      ; Address to copy the data to
NumBytes        DD     ?      ; Number of bytes to copy

PUSH   Source                 ; Push the parameters
PUSH   Destination            ;
PUSH   NumBytes               ;

CALL   VDHCopyMem             ; Call the function
                              ; Notice that it isn't necessary to pop the variable that was
                              ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
Source	DD	Address of the source data to be copied
Destination	DD	Address to copy the data to
NumBytes	DD	Number of bytes to copy.

Results

Success If the function is successful, it returns nothing.

Failure If NumBytes is *0, a system halt will result.*

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: Protection faults are handled as if a user application had performed the access. Therefore, writing to an invalid page causes a virtual device driver page fault handler to gain control. If there is no virtual device driver fault handler, a *don't care* page is mapped in at the faulting linear address.

Notice that touching invalid pages is allowed only in the context of the DOS Session. A virtual device driver cannot touch invalid pages through the HVDM (DOS Session Handle) alias region if it is not in the context of that DOS Session. A virtual device driver that violates this rule *causes a system halt.*

This function yields the CPU if it is necessary to ensure that the thread dispatch latency limit is not exceeded. This function also supports overlapping linear regions. However, the results are undefined when copying between regions with aliases (through VDHMapPages) to the same physical page.

If necessary, VDHLockMem is used to ensure that the status of the source and destination linear ranges does not change. This is required only if the source and destination ranges are in private DOS Session memory (accessed through the HVDM), and the call to VDHCopyMem is made in the context of some other process.

VDHCreateBlockPool

This function creates a pool of memory blocks of a specified size. The blocks are allocated from the system area so that the block addresses are valid in all process contexts.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHCreateBlockPool:NEAR
BlockSize      DD     ?       ; Size of block in bytes
OptionFlag     DD     ?       ; Allocation options bit flag

PUSH   BlockSize              ; Push the parameters
PUSH   OptionFlag             ;

CALL   VDHCreateBlockPool     ; Call the function
                              ; Notice that it isn't necessary to pop the variable that was
                              ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
BlockSize	DD	Size of block to initialize (in bytes).
OptionFlag	DD	Allocation options bit flag. Possible value: **VDHCBP_SWAPPABLE** Blocks are allocated from the swappable heap. Otherwise, blocks are allocated from the fixed heap and can be accessed at hardware interrupt time.

Results

Success If the function is successful, it returns a handle to the pool of memory blocks.

Failure If the function fails, it returns *0*. VDHGetError should be called to determine the nature of the problem. If OptionFlag is invalid, or if BlockSize is equal to *0*, *a system halt occurs*.

Remarks:

Context Issues: This function can be called in the initialization or task context.

DOS Session Terminations: Any block pools allocated by a virtual device driver for use by the terminating DOS Session is freed by using VDHDestroyBlockPool.

VDHCreateSel

This function creates a GDT selector that maps a linear range. This service is used to create a 16:16 pointer, which is passed to a 16-bit physical device driver.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHCreateSel:NEAR
LinearAddress     DD     ?    ; Linear address of the start of the buffer
BufferSize        DD     ?    ; Size of the buffer in bytes

PUSH   LinearAddress          ; Push the parameters
PUSH   BufferSize             ;

CALL   VDHCreateSel           ; Call the function
                              ; Notice that it isn't necessary to pop the variable that was
                              ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
LinearAddress	DD	Linear address of the beginning of the buffer.
BufferSize	DD	Size of the buffer in bytes. The expected range is *0*–*65535* with *0* interpreted as 65536.

Results

Success If the function is successful, it returns a non-zero selector value such that SELECTOR:0 points to the first byte of the linear range.

Failure If BufferSize is greater than 65535 or if there are no more selectors available, *a system halt occurs.*

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: Any selectors created to map memory in the terminating DOS Session must be destroyed with VDHDestroySel. The virtual device driver must ensure that any physical device driver that was given these selectors is finished with them.

Notes: The linear address must be in system memory. To create a 16:16 pointer to a DOS Session address (in the *0* to 1MB+64KB range) use the HVDM (DOS Session Handle) alias linear address. VDHDestroySel must be called when finished with this pointer.

VDHCreateSem

This function is used to create an event or mutex semaphore.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHCreateSem:NEAR
SemHandlePointer   DD    ?     ; Pointer to a semaphore handle
SemType            DD    ?     ; Type of semaphore of create

PUSH   SemHandlePointer        ; Push the parameters
PUSH   SemType                 ;

CALL   VDHCreateSem            ; Call the function
                               ; Notice that it isn't necessary to pop the variable that was
                               ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
SemHandlePointer	DD	Semaphore handle address.
SemType	DD	Type of semaphore to create. Values are: **VDH_EVENTSEM** Event semaphore **VDH_MUTEXSEM** Mutex semaphore.

Results

Success If the function is successful, it returns a non-zero value and places a handle to the semaphore in SemHandlePointer.

Failure If the function fails, it returns 0. VDHGetError should be called to determine the nature of the problem. An invalid semaphore type *causes a system halt*.

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: These semaphores are global and are not attached to a specific DOS Session. If a virtual device driver maintains semaphores on a per-DOS Session basis, it destroys the semaphores by using VDHDestroySem when the DOS Session (they are associated with) is terminated.

VDHDecodeProperty

This function decodes the specified format of a property string as shown below:

```
String Syntax = "N1-N2,N3-N4,N5,N6-N7,... "
```

Notice that N*x* are valid numbers in a specified base.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHDecodeProperty:NEAR
ppProperty          DD     ?     ; Pointer to a pointer to property string
StartNumberPtr      DD     ?     ; Location to return the starting number
EndNumberPtr        DD     ?     ; Location to return the ending number
BaseFlag            DD     ?     ; Flag to indicate the base to use

PUSH   ppProperty                ; Push the parameters
PUSH   StartNumberPtr            ;
PUSH   EndNumberPtr              ;
PUSH   BaseFlag                  ;

CALL   VDHDecodeProperty         ; Call the function
                                 ; Notice that it isn't necessary to pop the variable that was
                                 ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
ppProperty	DD	A pointer to a pointer to the property string to be decoded
StartNumberPtr	DD	Location for storing the returned starting number
EndNumberPtr	DD	Location for storing the returned ending number
BaseFlag	DD	Flag indicating which numeric base to use: **VDH_DP_DECIMAL** Decimal **VDH_DP_HEX** Hex.

Results

Success — If the function is successful, it returns a non-zero value and the ppProperty pointer is modified to point to the next range in the property string. StartNumberPtr and EndNumberPtr will have the range values. ppProperty is returned NULL after processing the last range.

Failure — If the function fails, it returns 0, indicating an invalid parameter. If BaseFlag contains an invalid value or if ppProperty is NULL, *a system halt occurs*.

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: If a single value is found, EndNumberPtr is *–1*. All the numbers are decoded in the specified base. A negative sign is used only as range delimiter. Numbers cannot have more than 8 digits or a failure is returned. If this service returns failure, it is still possible that the contents of StartNumberPtr or EndNumberPtr might have changed.

Because pointers passed are from a virtual device driver (not a user), there is no verification of addresses. Bad pointers result in page faults.

VDHDestroyBlockPool

This function releases a memory block pool.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHDestroyBlockPool:NEAR
PoolHandle     DD     ?      ; Handle to the memory block pool to release

PUSH   PoolHandle            ; Push the parameter

CALL   VDHDestroyBlockPool   ; Call the function
                             ; Notice that it isn't necessary to pop the variable that was
                             ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
PoolHandle	DD	Handle to the memory block pool to release.

Results

Success If the function was successful, it returns nothing.

Failure An invalid memory block pool handle *causes a system halt*.

Remarks:

Context Issues: This function can be called in the initialization or task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHDestroySel

This function destroys a GDT selector created by VDHCreateSel.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHDestroySel:NEAR
GDTSelector     DW     ?     ; GDT selector to destroy

PUSH   GDTSelector           ; Push the parameter

CALL   VDHDestroySel         ; Call the function
                             ; Notice that it isn't necessary to pop the variable that was
                             ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
GDTSelector	DW	GDT selector to be destroyed.

Results

Success If the function is successful, it returns nothing.

Failure If GDTSelector was not created by VDHCreateSel, *a system halt occurs*.

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: All selectors created with VDHCreateSel that map the terminating DOS Session must be destroyed.

VDHDestroySem

This function destroys a semaphore previously created with VDHCreateSem. The semaphore must be posted or unowned before calling this service.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHDestroySem:NEAR
SemHandle     DD     ?       ; Semaphore handle

PUSH   SemHandle             ; Push the parameter

CALL   VDHDestroySem         ; Call the function
                             ; Notice that it isn't necessary to pop the variable that was
                             ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
SemHandle	DD	Handle of the semaphore to destroy.

Results

Success If the function is successful, it returns nothing.

Failure If SemHandle is invalid or if the semaphore is *reset* or *owned, a system halt occurs.*

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHDevBeep

This function performs the preempt beep request on behalf of the requesting virtual device driver.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHDevBeep:NEAR
Frequency      DD     ?      ; Frequency of the beep in hertz
Duration       DD     ?      ; Duration of the beep in milliseconds

PUSH   Frequency             ; Push the parameters
PUSH   Duration              ;

CALL   VDHDevBeep            ; Call the function
                             ; Notice that it isn't necessary to pop the variable that was
                             ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
Frequency	DD	Frequency of the beep in hertz
Duration	DD	Duration of the beep in milliseconds.

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, it returns 0.

Remarks:

Context Issues: This function can be called in the task or interrupt context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHDevIOCtl

This function allows a virtual device driver to send device-specific commands to a physical device driver through the generic IOCtl interface. The parameters and error return codes are identical to the ones found in the DosDevIOCtl API. Refer to the *OS/2 2.0 Control Program Programming Reference* for a complete description of each DosDevIOCtl API parameter.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHDevIOCtl:NEAR
DevHandle       DD   ?    ; Device handle return by VDHOpen or VDHPhysicalDisk
Category        DD   ?    ; Category of function performed
Function        DD   ?    ; Function within category performed
ParmList        DD   ?    ; Address of application input parameter area
ParmLengthMax   DD   ?    ; Maximum size of the application input parameter area, in bytes
ParmLengthInOut DD   ?    ; Pointer to length (in bytes) of the parameters passed
DataArea        DD   ?    ; Pointer to the data area
DataLengthMax   DD   ?    ; Maximum size of the parameters passed, in bytes
DataLengthInOut DD   ?    ; Pointer to length (in bytes) of the parameters passed

PUSH   DevHandle          ; Push the parameters
PUSH   Category           ;
PUSH   Function           ;
PUSH   ParmList           ;
PUSH   ParmLengthMax      ;
PUSH   ParmLengthInOut    ;
PUSH   DataArea           ;
PUSH   DataLengthMax      ;
PUSH   DataLengthInOut    ;

CALL   VDHDevIOCtl        ; Call the function
                          ; Notice that it isn't necessary to pop the variable that was
                          ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
DevHandle	DD	Device handle return by VDHOpen or VDHPhysicalDisk
Category	DD	Category of function performed
Function	DD	Function within category performed
ParmList	DD	Pointer to the command-specific input parameter area
ParmLengthMax	DD	Maximum size of the application input parameter area, in bytes. ParmLengthInOut can be longer than this on input but not on output.
ParmLengthInOut	DD	Pointer to the length (in bytes) of the parameters passed by the caller in ParmList.
DataArea	DD	Pointer to the data area.
DataLengthMax	DD	Maximum size of the application data area, in bytes.
DataLengthInOut	DD	Pointer to the length (in bytes) of the parameters passed by the caller in ParmList. ParmLengthInOut can be longer than this on input but not on output.

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, it returns 0. VDHGetError should be called to determine the nature of the problem. VDHGetError can return the following errors:

ERROR_INVALID_HANDLE
ERROR_INVALID_FUNCTION
ERROR_INVALID_CATEGORY
ERROR_INVALID_DRIVE
ERROR_INVALID_PARAMETER.

If the function is called when not at DOS Session-task time, *a system halt occurs*.

Remarks:

Context Issues: This function can be called in the task or interrupt context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: Addresses (pointers) inside of device-specific Data or Parameter Packets are not translated.

VDHDisarmTimerHook

This function cancels a timer that was installed by VDHArmTimerHook before the handler was called.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHDisarmTimerHook:NEAR
TimerHook     DD     ?          ; Timer hook handle

PUSH   TimerHook                ; Push the parameter

CALL   VDHDisarmTimerHook       ; Call the function
                                ; Notice that it isn't necessary to pop the variable that was
                                ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
TimerHook	DD	Handle to the timer hook to disarm.

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, it returns 0. VDHGetError should be called to determine the nature of the problem. If TimerHook is invalid, *a system halt occurs.*

Remarks:

Context Issues: This function can be called in the initialization, task, or interrupt context.

DOS Session Terminations: Any timer hooks allocated in the context of the terminating DOS Session are deallocated automatically by the DOS Session Manager at DOS Session termination.

VDHEndUseVPMStack

This function must be called once with every call made to VDHBeginUseVPMStack. It switches back to the original DOS Session protect-mode stack when the use count goes to zero.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHEndUseVPMStack:NEAR

CALL   VDHEndUseVPMStack      ; Call the function
                              ; Notice that it isn't necessary to pop the variable that was
                              ; pushed; the Virtual DevHlp function does this.
```

Parameters: None.

Results: None.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHEnumerateVDMs

This function is used to run a worker function for each DOS Session in the system. The worker function can stop the enumeration by returning 0.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHEnumerateVDMs:NEAR
WorkerFcn      DD     ?      ; Worker function
FcnData        DD     ?      ; Function-specific data to be passed each call

PUSH   WorkerFcn             ; Push the parameters
PUSH   FcnData               ;

CALL   VDHEnumerateVDMs      ; Call the function
                             ; Notice that it isn't necessary to pop the variable that was
                             ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
WorkerFcn	DD	Worker function that is the routine that does the work
FcnData	DD	Function-specific data to be passed each call.

The worker function uses the following interface:

```
EnumHook  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 8] - hVDM   - Handle to DOS Session
;           [ESP + 4] - ulData - Function-specific data
;    EXIT   SUCCESS - Return a non-zero value in EAX, enumeration continues
;           FAILURE - Return 0 in EAX, stop enumeration
```

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, it returns 0 indicating that a worker function stopped the enumeration.

Remarks:

Context Issues: This function can be called in the task and interrupt context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHExchangeMem

This function exchanges the contents of two regions of linear address space. Overlapping regions are not supported.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHExchangeMem:NEAR
Source          DD     ?      ; Source address
Destination     DD     ?      ; Destination address
NumBytes        DD     ?      ; Number of bytes to be exchanged

PUSH   Source                 ; Push the parameters
PUSH   Destination            ;
PUSH   NumBytes               ;

CALL   VDHExchangeMem         ; Call the function
                              ; Notice that it isn't necessary to pop the variable that was
                              ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
Source	DD	Address of the source data
Destination	DD	Address of the target region
NumBytes	DD	Number of bytes to exchange.

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, it returns 0. VDHGetError should be called to determine the nature of the problem.

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: This function fails, if the two memory regions to be exchanged overlap. Protection faults are handled as if a user application had performed the access. Therefore, writing to read-only pages or invalid page-table entries causes a virtual device driver page fault handler to gain control or causes the default behavior (that is, DOS Session termination). This function also yields the CPU, if it is necessary to ensure that the thread dispatch latency limit is not exceeded.

VDHFindFreePages

This function starts at a specified linear address and searches linear memory (up to a specified limit) for the largest free linear memory block. The address and size of the largest region found are returned.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHFindFreePages:NEAR
StartAddress        DD     ?     ; Starting address
Pages               DD     ?     ; Size of the range to search (in pages)

PUSH   StartAddress               ; Push the parameters
PUSH   Pages                      ;

CALL   VDHFindFreePages           ; Call the function
                                  ; Notice that it isn't necessary to pop the variable that was
                                  ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
StartAddress	DD	Starting address of the linear memory range to be searched. Must be page-aligned and between *0* and 110000H (1MB+64KB).
Pages	DD	On input: The size of the linear memory range to search, in pages. On output: The size of the region found, in pages.

Results

Success — If the function is successful, it returns the address of the region found and sets Pages to the size of the region found (in pages).

Failure — If the function fails, it returns *0*. VDHGetError should be called to determine the nature of the problem.

Remarks:

Context Issues: This function can be called in the initialization context where it searches only the global linear map. This function can also be called in DOS Session-task context (DOS Session creation only), where it searches only the DOS Session's private linear map.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHFreeBlock

This function releases a memory block previously allocated by VDHAllocBlock. The memory block is returned to the user's memory block pool for reuse.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHFreeBlock:NEAR
PoolHandle      DD     ?      ; Handle to a memory block pool
BlockToFree     DD     ?      ; Pointer to the memory block to free

PUSH   PoolHandle             ; Push the parameters
PUSH   BlockToFree            ;

CALL   VDHFreeBlock           ; Call the function
                              ; Notice that it isn't necessary to pop the variable that was
                              ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
PoolHandle	DD	Handle (from VDHCreateBlockPool) to the pool of memory blocks that contains the memory block to free
BlockToFree	DD	Address of the memory block to free.

Results

Success If the function is successful, it returns nothing.

Failure If BlockPool is not a valid handle to a memory block pool, or if BlockToFree is not a block allocated from the BlockPool memory block pool, *a system halt occurs.*

Remarks:

Context Issues: This function can be called in the initialization or task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: The only way to reclaim memory for freed blocks is to call VDHDestroyBlockPool.

VDHFreeDMABuffer

This function frees a DMA buffer previously allocated by VDHAllocDMABuffer.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHFreeDMABuffer:NEAR
LinearAddress     DD     ?     ; Starting linear address of the DMA buffer

PUSH   LinearAddress           ; Push the parameter

CALL   VDHFreeDMABuffer        ; Call the function.
                               ; Notice that it isn't necessary to pop the variable that was
                               ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
LinearAddress	DD	Starting linear address of the DMA buffer.

Results

Success If the function is successful, it returns nothing.

Failure If LinearAddress is invalid, *a system halt occurs.*

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHFreeHook

This function disarms and frees a hook.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHFreeHook:NEAR
HookHandle     DD     ?      ; Hook handle from VDHAllocHook

PUSH   HookHandle            ; Push the parameter

CALL   VDHFreeHook           ; Call the function
                             ; Notice that it isn't necessary to pop the variable that was
                             ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
HookHandle	DD	Hook handle (from VDHAllocHook) for the hook to free.

Results

Success If the function is successful, it returns nothing.

Failure If HookHandle is invalid, *a system halt occurs*.

Remarks:

Context Issues: This function can be called in the initialization or DOS Session-task context.

DOS Session Terminations: Any context hooks allocated in the context of the terminating DOS Session are deallocated automatically by the DOS Session Manager.

VDHFreeMem

This function frees memory that was previously allocated by VDHAllocMem.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHFreeMem:NEAR
MemAddress     DD     ?     ; Address of the memory block to be freed

PUSH   MemAddress           ; Push the parameter

CALL   VDHFreeMem           ; Call the function
                            ; Notice that it isn't necessary to pop the variable that was
                            ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
MemAddress	DD	Address of the memory block to be freed. Pointer is originally obtained from VDHAllocMem.

Results

Success If the function is successful, it returns nothing.

Failure If MemAddress is not an address pointer allocated by VDHAllocMem, *a system halt occurs.*

Remarks:

Context Issues: This function can be called in the initialization or task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHFreePages

This function frees a memory object. All the memory associated with the memory object is released.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHFreePages:NEAR
ObjectAddress     DD     ?     ; Address of the memory object to release.

PUSH   ObjectAddress           ; Push the parameter

CALL   VDHFreePages            ; Call the function
                               ; Notice that it isn't necessary to pop the variable that was
                               ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
ObjectAddress	DD	Address of the memory object to be freed.

Results

Success If the function is successful, it returns nothing.

Failure If ObjectAddress was not allocated by VDHAllocPages or VDHReallocPages, *a system halt occurs*.

Remarks:

Context Issues: This function can be called in the initialization or task context. At initialization time, only global memory can be freed.

DOS Session Terminations: Any allocations made in the terminating DOS Sessions private area, or on behalf of the terminating DOS Session, must be released by using VDHFreePages.

Notes: This function succeeds only if the linear range specified was allocated by a previous call to VDHAllocPages or VDHReallocPages. Freeing a memory region that was not allocated causes an internal error (a probable *system halt*).

If the region, starting at ObjectAddress, has been broken into two or more regions by calls to VDHMapPages, then VDHFreePages is not used.

VDHFreezeVDM

This function freezes a DOS Session, which prevents it from executing. The specified DOS Session is not allowed to execute any V86-mode code until VDHThawVDM is called. This freeze occurs when the specified DOS Session leaves kernel mode. The DOS Session does not execute any V86-mode code from the time VDHFreezeVDM is called until the matching VDHThawVDM is called.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHFreezeVDM:NEAR
VDMHandle     DD     ?       ; Handle of the DOS Session to freeze

PUSH   VDMHandle             ; Push the parameter

CALL   VDHFreezeVDM          ; Call the function
                             ; Notice that it isn't necessary to pop the variable that was
                             ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
VDMHandle	DD	Handle to the DOS Session to freeze.

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, it returns 0. VDHGetError should be called to determine the nature of the problem. If VDMHandle is not a valid DOS Session handle, *a system halt occurs.*

Remarks:

Context Issues: This function can be called in the task or interrupt context.

DOS Session Terminations: The DOS Session Manager *thaws* a frozen DOS Session prior to calling any VDM_TERMINATE user hooks.

Notes: Each DOS Session has a *freeze count*. This count starts at zero when the DOS Session is created. Each VDHFreezeVDM call adds one to this count. Each VDHThawVDM call subtracts one from this count. The DOS Session is frozen when the freeze count is *1* or greater. The DOS Session is thawed when the freeze count is *0*. This allows multiple virtual device drivers to perform freeze and thaw operations without inadvertently causing the DOS Session to run. If this count exceeds MAX_FREEZE_COUNT, then VDHFreezeVDM returns VDHERR_FROZEN_LIMIT.

VDHGetCodePageFont

This function returns the address of the code page font information if code page support is active and a font is available for the given character cell dimensions.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHGetCodePageFont:NEAR
CellWidth       DD      ?       ; Width of character cells, in pels
CellHeight      DD      ?       ; Height of character cells, in pels
FontAddress     DD      ?       ; Address where font table addresses will be copied

PUSH   CellWidth                ; Push the parameters
PUSH   CellHeight               ;
PUSH   FontAddress              ;

CALL   VDHGetCodePageFont       ; Call the function
                                ; Notice that it isn't necessary to pop the variable that was
                                ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
CellWidth	DD	Width of character cells, measured in pels (pixels)
CellHeight	DD	Height of character cells, measured in pels (pixels)
FontAddress	DD	Address where font table addresses will be copied.

Results

Success If the function is successful, it returns the size of the font table in bytes.

Failure If the function fails, it returns *0*. VDHGetError should be called to determine the nature of the problem.

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: If this function was called on behalf of the terminating DOS Session, VDHReleaseCodePageFont must be called to release the fonts.

Notes: System memory can be allocated to hold the font, especially if it is a merged font (that is, merged from two separate font tables). When the virtual device driver is finished with the font, it must call VDHReleaseCodePageFont.

VDHGetDirtyPageInfo

This function returns a bit vector for the dirty pages of a specified DOS Session for a specified range of pages. The dirty bits for the range are reset.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHGetDirtyPageInfo:NEAR
VDMHandle           DD    ?   ; Handle to a DOS Session
StartingAddress     DD    ?   ; Address to start the scan
PageRange           DD    ?   ; Page limit to the length of the scan

PUSH   VDMHandle              ; Push the parameters
PUSH   StartingAddress        ;
PUSH   PageRange              ;

CALL   VDHGetDirtyPageInfo    ; Call the function
                              ; Notice that it isn't necessary to pop the variable that was
                              ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
VDMHandle	DD	Handle to the DOS Session that owns the object. A *0* value indicates the current DOS Session.
StartingAddress	DD	Starting address of the range of pages to scan.
PageRange	DD	Range (number of pages) to scan. Maximum allowed is 32.

Results

Success — If the function is successful, it returns the bit vector. The low-order bit (that is, bit 0, the rightmost bit) is the dirty bit for the first page scanned. Bit 1 (from the right) is the dirty bit for the next page, and so forth for all the pages scanned.

Failure — If any of the input parameters are invalid, *a system halt occurs*.

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: The virtual dirty bits remain set until cleared by this function, regardless of what swapping activity has occurred. The actual dirty bits in the page tables are reset whenever a page is swapped to the swap device and brought back into memory. This function is valid only for linear addresses up to 110000H (1MB+64KB) in a DOS Session.

VDHGetError

This function returns the error code from the last Virtual DevHlp service called.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHGetError:NEAR

CALL   VDHGetError        ; Call the function
                          ; Notice that it isn't necessary to pop the variable that was
                          ; pushed; the Virtual DevHlp function does this.
```

Parameters: None.

Results

Success If the function is successful, it returns the error code.

Failure If the function fails, it returns *0* indicating that the last Virtual DevHlp call did not have an error.

Remarks:

Context Issues: This function can be called in any context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: The return value is guaranteed to be valid as long as the virtual device driver does not block or yield between the Virtual DevHlp service that failed and the call to VDHGetError. VDHGetError and VDHSetError work at hardware interrupt time. Each level of interrupt nesting (where task time is level *0*), has a separate error code variable.

VDHGetSelBase

This function returns the base address for a Local Descriptor Table (LDT) selector.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHGetSelBase:NEAR
Selector        DD     ?
pBasePointer    DD     ?

PUSH   Selector              ; Push the parameters
PUSH   pBasePointer          ;

CALL   VDHGetSelBase         ; Call the function
                             ; Notice that it isn't necessary to pop the variable that was
                             ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
Selector	DD	Selector
pBasePointer	DD	Pointer for returned address.

Results

Success If the function is successful, it returns a non-zero value. The base address is returned in the variable pointed to by pBasePointer.

Failure If the function fails, it returns 0.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: Error checking is performed only to ensure that the selector is allocated.

VDHGetVPMExcept

This function gets the current value from the protect-mode exception table.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHGetVPMExcept:NEAR
Vector                    DD    ?
pHandlerAddressPointer    DD    ?
pFlag                     DD    ?

PUSH   Vector                      ; Push the parameters
PUSH   pHandlerAddressPointer      ;
PUSH   pFlag                       ;

CALL   VDHGetVPMExcept             ; Call the function
                                   ; Notice that it isn't necessary to pop the variable that was
                                   ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
Vector	DD	Exception vector number (*0* – 1FH)
pHandlerAddressPointer	DD	Pointer to put handler address
pFlag	DD	Pointer to a flag variable for returning the flag value. Possible values are: **VPMXCPT32** A 32-bit handler was registered **VPMXCPT_REFLECT** The exception is reflected back to a V86-mode handler.

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, it returns 0.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHGetVPMIntVector

This function gets the application Ring 3 handler in the protect-mode interrupt chain. This service is used only for DOS Protect Mode Interface (DPMI) support.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHGetVPMIntVector:NEAR
Vector                  DD   ?
HandlerAddressPointer   DD   ?

PUSH   Vector                   ; Push the parameters
PUSH   HandlerAddressPointer    ;

CALL   VDHGetVPMIntVector       ; Call the function
                                ; Notice that it isn't necessary to pop the variable that was
                                ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
Vector	DD	Interrupt vector number
HandlerAddressPointer	DD	Pointer for return of handler address.

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, it returns 0.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHHaltSystem

This function *causes a system halt.*

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHHaltSystem:NEAR

CALL   VDHHaltSystem       ; Call the function
                           ; Notice that it isn't necessary to pop the variable that was
                           ; pushed; the Virtual DevHlp function does this.
```

Parameters: None.

Results: None.

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: Calling VDHHaltSystem is an extremely drastic action. Calling VDHPopup and VDHKillVDM is the preferred method for handling a problem. VDHHaltSystem is used only if the virtual device driver is certain that the entire Multiple DOS Sessions environment is inoperable. In general, this function is used only by a base virtual device driver (for example, the virtual Video or Keyboard device driver) whose absence would render the Multiple DOS Sessions environment unusable. VDHPopup is called prior to VDHHaltSystem to tell the user what has happened.

VDHHandleFromPID

This function returns the DOS Session handle for a given Process ID.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHHandleFromPID:NEAR
ProcessID     DW     ?      ; A DOS Session process ID

PUSH   ProcessID            ; Push the parameter

CALL   VDHHandleFromPID     ; Call the function
                            ; Notice that it isn't necessary to pop the variable that was
                            ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
ProcessID	DW	A DOS Session Process ID.

Results

Success If the function is successful, it returns the DOS Session handle for the Process ID.

Failure If the function fails, it returns *0*. VDHGetError should be called to determine the nature of the problem.

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHHandleFromSGID

This function determines the DOS Session handle given the Screen Group ID.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHHandleFromSGID:NEAR
ScreenGroupID     DW     ?    ; Screen Group ID

PUSH   ScreenGroupID          ; Push the parameter

CALL   VDHHandleFromSGID      ; Call the function
                              ; Notice that it isn't necessary to pop the variable that was
                              ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
ScreenGroupID	DW	Screen Group ID.

Results

Success If the function is successful, it returns a handle to the DOS Session that was given the Screen Group ID.

Failure If the function fails, it returns *0*. VDHGetError should be called to determine the nature of the problem.

Remarks:

Context Issues: This function can be called in the task or interrupt context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHInstallFaultHook

This function sets a page fault handler for a region of linear address space. The page fault handler receives control when a DOS Session touches a page in a reserved region that was invalidated by VDHMapPages. The handler does not get control, if the page was indicated *Not Present* by the Page Manager.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHInstallFaultHook:NEAR
VDMHandle               DD    ?      ; DOS Session handle; 0 = current DOS Session
StartingAddress         DD    ?      ; Starting linear address
Pages                   DD    ?      ; Number of pages
PageFaultHandlerFcnPtr  DD    ?      ; Page fault handler function
OptionFlag              DD    ?      ; Options bit flag

PUSH   VDMHandle                     ; Push the parameters
PUSH   StartingAddress               ;
PUSH   Pages                         ;
PUSH   PageFaultHandlerFcnPtr        ;
PUSH   OptionFlag                    ;

CALL   VDHInstallFaultHook           ; Call the function
                                     ; Notice that it isn't necessary to pop the variable that was
                                     ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
VDMHandle	DD	DOS Session Handle. If the parameter contains a *0*, use the currently active DOS Session.
StartingAddress	DD	Starting linear address.
Pages	DD	Number of pages.
PageFaultHandlerFcnPtr	DD	Page fault handler function. See below.
OptionFlag	DD	Options bit flag. See below.

OptionFlag Options bit flag. Possible value:

VDHIFH_ADDR If set, pVDM (see below) is a byte-granular address. Otherwise, pVDM is a page-granular address.

The PageFaultHandlerFcnPtr function pointer contains the address of a function with the following interface:

```
PageFaultHandler  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 4] - pVDM - Address of the page in which fault occurred
;    EXIT   None
;
; CONTEXT   DOS Session-task.
```

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, it returns 0. VDHGetError should be called to determine the nature of the problem. If any of the input parameters are invalid, *a system halt occurs.*

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: Any fault hooks for the terminating DOS Session must be released by using VDHRemoveFaultHook.

Notes: In order for the fault handler to terminate the DOS Session, it must call VDHKillVDM and return. Page fault hooks are stored on a per-DOS Session basis.

VDHInstallIntHook

This function sets a handler for a V86 interrupt. The virtual device driver's interrupt handler gets control after any subsequent hooked DOS interrupt handlers.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHInstallIntHook:NEAR
Reserved        DD     0       ; Reserved; must be set to 0
Vector          DD     ?       ; Number of the interrupt vector to hook
HookFcn         DD     ?       ; Address of the hook routine
OptionFlag      DD     ?       ; Interface options flag

PUSH   Reserved                ; Push the parameters
PUSH   Vector                  ;
PUSH   HookFcn                 ;
PUSH   OptionFlag              ;

CALL   VDHInstallIntHook       ; Call the function
                               ; Notice that it isn't necessary to pop the variable that was
                               ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
Reserved	DD	Reserved. Must be set to *0*.
Vector	DD	Number of the interrupt vector to hook (0 – 255).
HookFcn	DD	Address of the hook routine.
OptionFlag	DD	Interface options flag. See below.

OptionFlag Interface options flag. Possible values:

VDH_ASM_HOOK Use assembly language hook interface. Otherwise, use the calling convention in the function prototype VDHENTRY found in the MVDM.INC include file, which included with the Toolkit.

VDH_PRE_HOOK If set, install handler as a pre-reflection hook, that is, call the hook routine before reflecting the interrupt to V86 mode.

Interface for the interrupt hook routine:

```
HookRoutine  PROC  NEAR
; PARAMETERS
;    ENTRY  EBX - pcrf - Client register frame pointer
;    EXIT   If carry flag set, chain to next virtual device driver
;           If carry flag clear, don't chain to the next virtual device driver
;
; USES      EAX, ECX, EDX, FS, GS, Flags
; CONTEXT   DOS Session-task.
```

Results

Success If the function was successful, it returns a non-zero value.

Failure If the function fails, it returns 0. VDHGetError should be called to determine the nature of the problem. If HookFcn is invalid or if Vector is out of range, *a system halt occurs*.

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: There are no DOS Session termination implications for this function. Interrupt hooks are automatically removed during DOS Session termination.

Notes: The effect of this service is per-DOS Session; the software interrupt handler must be installed as each DOS Session is created. The return value from the interrupt hook controls whether the next virtual device driver is chained to or not. An explicit VDHPopInt is needed to remove the software interrupt (so that the ROM code is not executed).

VDHInstallIOHook

This function is used to install I/O port hooks.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHInstallIOHook:NEAR
Reserved          DD    0     ; Reserved; must be set to 0
StartingPort      DD    ?     ; Starting port number
NumPorts          DD    ?     ; The number of ports from the starting point
IOPortHookEntry   DD    ?     ; Pointer to I/O port hook entry
Flag              DD    ?     ; Interface and trapping options

PUSH   Reserved               ; Push the parameters
PUSH   StartingPort           ;
PUSH   NumPorts               ;
PUSH   IOPortHookEntry        ;
PUSH   Flag                   ;

CALL   VDHInstallIOHook       ; Call the function
                              ; Notice that it isn't necessary to pop the variable that was
                              ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
Reserved	DD	Reserved. Must be set to *0*.
StartingPort	DD	Number of the starting port.
NumPorts	DD	The number of ports from the starting port.
IOPortHookEntry	DD	Pointer to I/O port hook entry.
Flag	DD	See below.

Flag Indicates interface and trapping options. Possible values:

VDHIIH_ASM_HOOK Use assembler language hook interface (register-based). Otherwise, use a C hook interface (stack-based).

VDHIIH_ALWAYS_TRAP Always trap this range of addresses. The virtual device driver cannot call VDHSetIOHookState on any ports in the range.

VDHIIH_NO_SIMULATE Does not change WORD or other I/O handlers to simulation routines; allows WORD handlers for 1-port ranges.

VDHIIH_IGNORE Sets a system handler that does the following:

On WRITES: The OUT succeeds, but no actual I/O is performed.
On READS: The IN succeeds, and always returns −1 (all bits set to *1*).

This is also the default behavior of unhooked ports. This behavior is compatible with the behavior of a machine when there is no hardware on a particular I/O address.

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, it returns 0. VDHGetError should be called to determine the nature of the problem. If IOPortHookEntry is invalid or if StartingPort or NumPorts are out of range, *a system halt occurs.*

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: There are no DOS Session termination implications for this function. I/O hooks are automatically removed during DOS Session termination.

Notes: This function installs I/O port trap handlers for a contiguous range of I/O addresses. Trapping is initially enabled for the specified ports. The I/O hook entry contains the handlers for each kind of I/O operation. If an entry in the table is NULL, that I/O type is simulated by using a lower I/O type (for example, WORD-I/O is simulated by performing two byte-I/O operations). I/O port hooks are kept on a per-DOS Session basis; a virtual device driver generally installs I/O hooks at DOS Session creation time.

VDHSetIOHookState can be used to enable or disable I/O trapping on a per-port, per-DOS Session basis except when the VDHIIH_ALWAYS_TRAP option is specified. In this case, VDHSetIOHookState cannot be called on any port in the range for the current DOS Session.

I/O hooks can take their parameters either from registers or the stack. High performance hooks take register parameters (the I/O hook router is optimized for this case). Hooks with lower performance requirements take stack parameters, and therefore can be written in C.

WordInput, WordOutput and Other hooks can be NULL. In this case, the DOS Session Manager simulates these routines by using the ByteInput and ByteOutput routines. If the ByteInput or ByteOutput handlers are NULL, the DOS Session Manager uses default internal routines that do input and output to the hardware ports. This is most useful when a virtual device driver shadows the hardware state (by trapping OUT instructions) but does not care what is returned by IN instructions.

The client register frame (CRF) is updated as the instructions are simulated. For example, when the string I/O instructions are simulated, the client's EDI/ESI and ECX is incremented or decremented.

In a particular DOS Session, a particular I/O port address can be hooked only by a single virtual device driver. If this function fails, no I/O hooks are installed.

The routing of I/O instructions to the supplied port hooks covers the specified range exactly. If an I/O instruction overlaps port address ranges hooked on separate VDHInstallIOHook calls, the instruction is simulated with smaller I/O operations to ensure that the hooks in each range are called correctly. For example, a virtual device driver that hooks 100H – 103H, and installs a byte, WORD, and DWORD (other) handler would work as follows:

```
VDHInstallIOHook (reserved,0x100,4,&iohA,NULL);
```

The I/O instructions to each address with the length indicated cause the following hooks to be called:

Address	Length	Hooks Called
100	4	iohA.Other(100,DWORD)
101	4	iohA.Word(101) iohA.Byte(103)
102	4	iohA.Word(102)
103	4	iohA.Byte(103)

VDHRemoveIOHook must be called with identical StartingPort and NumPorts in order to remove these I/O hooks. Port hooks cannot be removed for a subset of the range of ports hooked on the call to VDHInstallIOHook. If the IOPortHookEntry is in instance data, the address passed to VDHInstallIOHook must be the same address passed to VDHRemoveIOHook or VDHSetIOHookState.

Virtual device drivers hook ports in all cases even if the desired behavior is to act as though direct port access were allowed. This keeps the I/O Permissions Map small. Only a port that requires high performance (for example, a graphics card) is allowed to go straight through. The existence of such ports increases the size of the I/O Permissions Map in the Task State segment even if SetIOHookState is never called for them.

Interfaces for IOPortHookEntry are as follows:

```
ioh STRUC
    ioh_pbihByteInput    DD   ?
    ioh_pbohByteOutput   DD   ?
    ioh_pwihWordInput    DD   ?
    ioh_pwohWordOutput   DD   ?
    ioh_pothOther        DD   ?
ioh ENDS

ByteInput  PROC  NEAR
; PARAMETERS
;    ENTRY  EDX - port - Port number
;           EBX - pcrf - Client register frame pointer
;    EXIT   Data read returned in AL
;
; USES      EAX, ECX, EDX, FS, GS, Flags
; CONTEXT   DOS Session-task

ByteOutput  PROC  NEAR
; PARAMETERS
;    ENTRY  AL  - bOutputData - Data to write
;           EBX - pcrf        - Client register frame pointer
;           EDX - port        - Port number
;    EXIT   None
;
; USES      EAX, ECX, EDX, FS, GS, Flags
; CONTEXT   DOS Session-task

WordInput  PROC  NEAR
; PARAMETERS
;    ENTRY  EBX - pcrf - Client register frame pointer
;           EDX - port - Port number
;    EXIT   Data read returned in AX
;
; USES      EAX, ECX, EDX, FS, GS, Flags
; CONTEXT   DOS Session-task
```

```
WordOutput  PROC  NEAR
; PARAMETERS
;    ENTRY  AX  - usOutputData - Data to write
;           EDX - port         - Port number
;           EBX - pcrf         - Client register frame pointer
;    EXIT   None
;
; USES      EAX, ECX, EDX, FS, GS, Flags
; CONTEXT   DOS Session-task


OtherHook  PROC  NEAR
; PARAMETERS
;    ENTRY  EAX - ulOutputData         - Data for DWORD writes
;           [ESP + 16] - pulInputData  - Pointer to put input data
;                        (See NOTE, below)
;           EDX - port                 - Port number
;           ECX - flType               - I/O type
;               IO_TYPE_SHIFT    - Shift for type
;               IO_TYPE_MASK     - Mask for type
;               IO_TYPE_INPUT    - If set input else output
;               IO_TYPE_BYTE     - If set byte else WORD/DWORD
;               IO_TYPE_DWORD
;               IO_TYPE_STRING
;               IO_TYPE_REVERSE
;               IO_TYPE_ADDR32
;               IO_TYPE_REP
;               IO_SEG_SHIFT     - Shift for SEG  field
;               IO_SEG_MASK      - Mask for SEG field
;               IO_SEG_CS
;               IO_SEG_SS
;               IO_SEG_ES
;               IO_SEG_FS
;               IO_SEG_GS
;           EBX - pcrf                 - Client register frame pointer
;    EXIT   If assembly language hook,
;              EAX - Data from DWORD reads
;              If carry flag set, simulate the I/O operation
;              If carry flag clear, I/O is done
;           If C hook
;              *pulInputData - Data from DWORD reads
;              If returns 0 in EAX, simulate the I/O operation
;              If returns a non-zero value in EAX, I/O is done
;
; USES      EAX, ECX, EDX, FS, GS, Flags
; CONTEXT   DOS Session-task
; NOTES     The ASM OtherHook hook does not need the pulInputData parameter
;           because it returns DWORD input values in EAX, with the
;           carry flag used to indicate done/simulate.
;           The C hook has to use EAX to return status, and so needs the
;           additional parameter.
```

VDHInstallUserHook

This function is used to set a handler for a specific DOS Session event.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHInstallUserHook:NEAR
Event        DD     ?       ; A DOS Session event for which a handler is installed
UserHook     DD     ?       ; User's handler for this event

PUSH   Event                ; Push the parameters
PUSH   UserHook             ;

CALL   VDHInstallUserHook   ; Call the function
                            ; Notice that it isn't necessary to pop the variable that was
                            ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
Event	DD	A DOS Session event. See below.
UserHook	DD	A user-defined handler for the event.

Event A DOS Session event (such as DOS Session termination). Possible values are:

VDD_EXIT	DOS Session support is shutting down
VDM_CREATE	DOS Session creation
VDM_TERMINATE	DOS Session termination
VDM_FOREGROUND	DOS Session to the foreground
VDM_BACKGROUND	DOS Session to the background
VDM_CREATE_DONE	DOS Session creation completed successfully
VDM_VDD_CREATE_DONE	DOS Session virtual device driver creation completed
VDM_PDB_DESTROY	DOS Program Data Block (PDB) destroyed in DOS Session
VDM_PDB_CHANGE	PDB changed in DOS Session
VDM_CODEPAGE_CHANGE	Code page change event
VDM_TITLE_CHANGE	DOS Session title change event
VDM_MEMORY_MAPPED_IN	Pages mapped into a DOS Session (*0* to 1MB+64KB)
VDM_MEMORY_UN_MAPPED	Pages unmapped from a DOS Session (*0* to 1MB+64KB).

The interfaces for the UserHook handlers (depending on the event that is handled) are:

```
pfnCreate  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 4] - hvdm - Handle to DOS Session
;    EXIT   SUCCESS  Return a non-zero value
;           FAILURE  Return 0
;                    DOS Session creation fails; the DOS Session Manager calls all the
;                    VDM_TERMINATE hooks for all virtual device drivers
;                    that returned SUCCESS on this DOS Session creation.
;
; CONTEXT   DOS Session-task
```

```
pfnTerminate  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 4] - hvdm - DOS Session being terminated
;    EXIT   None
;
; CONTEXT   DOS Session-task

pfnForeground  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 4] - hvdm - DOS Session coming to the foreground
;    EXIT   None
;
; CONTEXT   Task

pfnBackground  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 4] - hvdm - DOS Session going to the background
;    EXIT   None
;
; CONTEXT   Task

pfnExit  PROC  NEAR
; PARAMETERS
;    ENTRY  None
;    EXIT   None
;
; CONTEXT   Initialization and Task

pfnCreateDone  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 4] - hvdm - Handle to DOS Session
;    EXIT   None
;
; CONTEXT   DOS Session Creation

pfnVDDCreateDone  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 4] - hvdm - Handle to DOS Session
;    EXIT   None
;
; CONTEXT   DOS Session Creation

pfnPDBDestroyed  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 8] - hvdm    - Handle to DOS Session
;           [ESP + 4] - segPDB  - V86 segment of terminating PDB
;    EXIT   None
;
; CONTEXT   DOS Session-task

pfnPDBSwitched  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 8] - hvdm   - Handle to DOS Session
;           [ESP + 4] - segPDB - V86 segment of new PDB
;    EXIT   None
;
; CONTEXT   DOS Session-task
```

```
pfnCodePageChanged  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 4] - ulCodePage - New code page
;    EXIT   SUCCESS  Return a non-zero value in EAX
;           FAILURE  Return 0 in EAX
;                    An error is returned on the set code-page operation, but
;                    the rest of the device-code-page handlers are called.
;
; CONTEXT   DOS Session-task

pfnVDMTitleChange  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 4] - pszTitle - new DOS Session Title
;    EXIT   SUCCESS  Return a non-zero value in EAX
;           FAILURE  Return 0 in EAX
;
; CONTEXT   DOS Session-task
; NOTES     1. This event is called for both full screen and windowed DOS Session.
;           2. If pszTitle is NULL, the virtual device driver should treat it
;              as the DOS Session's default and original title.
;           3. Ideally there should be only one virtual device driver for this
;              hook, but this is not a restriction.  One of the virtual device
;              drivers registered is responsible for putting the title.  Only
;              this virtual device driver returns a non-zero value, all others
;              return 0.

pfnMemoryMappedIn  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 16] - hvdm   - Handle to DOS Session
;           [ESP + 12] - page   - Page address
;           [ESP +  8] - cpages - Number of pages mapped in from
;                                                 the starting page address
;           [ESP +  4] - fl     - Type of mapping, (see VDHMapPages)
;    EXIT   SUCCESS  Return a non-zero value in EAX
;           FAILURE  Return 0 in EAX
;
; CONTEXT   DOS Session-task

pfnMemoryUnMapped  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 16] - hvdm   - Handle to DOS Session
;           [ESP + 12] - page   - Page address
;           [ESP +  8] - cpages - Number of pages unmapped from
;                                                 the starting page address
;           [ESP +  4] - fl     - Type of mapping, (see VDHMapPages)
;    EXIT   SUCCESS  Return a non-zero value in EAX
;           FAILURE  Return 0 in EAX
;
; CONTEXT   DOS Session-task
```

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, an invalid input parameter or a kernel heap overflow (that is, if the system runs out of memory) *causes a system halt*.

Remarks:

Context Issues: This function can be called only in the initialization context.

DOS Session Terminations: The VDM_TERMINATE hook is called at DOS Session termination. In the case of a partially created DOS Session, the VDM_TERMINATE handler is called only if all the registered VDM_CREATE handlers are called successfully. Partially created DOS Sessions might occur when, for example, the Create_Handler for Virtual Device Driver A returned successfully, but the Create_Handler for Virtual Device Driver B failed. In this case, the Terminate_Handler for Virtual Device Driver A will not be called. This should be taken into consideration when deciding what to do in a Create_Handler and a Create_Done_Handler.

VDHIsVDMFrozen

This function returns the freeze state of a DOS Session.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHIsVDMFrozen:NEAR
VDMHandle     DD     ?        ; Handle to the DOS Session

PUSH   VDMHandle              ; Push the parameter

CALL   VDHIsVDMFrozen         ; Call the function
                              ; Notice that it isn't necessary to pop the variable that was
                              ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
VDMHandle	DD	Handle to the DOS Session in question.

Results

Success If the DOS Session is not frozen, the function returns 0. Otherwise, the function returns a non-zero value indicating the DOS Session's *freeze count*.

Failure If VDMHandle is an invalid DOS Session handle, *a system halt occurs*.

Remarks:

Context Issues: This function can be called in the task or interrupt context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: See "VDHFreezeVDM" on page 5-52 for a full discussion of freeze counting.

VDHKillVDM

This function terminates (kills) the DOS Session at the earliest opportunity. V86 code is no longer executed in the context of the terminated DOS Session.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHKillVDM:NEAR
VDMHandle     DD     ?        ; DOS Session handle

PUSH   VDMHandle              ; Push the parameter

CALL   VDHKillVDM             ; Call the function
                              ; Notice that it isn't necessary to pop the variable that was
                              ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
VDMHandle	DD	Handle to the DOS Session to terminate. A value of *0* indicates the current DOS Session.

Results

Success If the function is successful, it returns nothing.

Failure If VDMHandle is not a valid DOS Session handle, *a system halt occurs.*

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: When a DOS Session is terminating, all virtual device drivers that registered a VDM_TERMINATE hook by using VDHInstallUserHook are called.

Notes: This function sets a flag and returns. If the calling virtual device driver is in the context of the DOS Session to be terminated, it must return to its caller in order for the termination to occur.

VDHLockMem

This function verifies that a specified region is accessible in the requested manner, and locks the memory in the requested manner.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHLockMem:NEAR
StartingLinAddr     DD    ?     ; The starting linear address of the region in the user process to be locked
NumBytes            DD    ?     ; The size of the region to lock, in bytes
OptionFlag          DD    ?     ; Access flags
PagelistArrayPtr    DD    ?     ; Pointer to location to store VDHPageList array
ArrayCountPtr       DD    ?     ; Points to location to store a count of VDHPageList_s elements returned

PUSH   StartingLinAddr          ; Push the parameters
PUSH   NumBytes                 ;
PUSH   OptionFlag               ;
PUSH   PagelistArrayPtr         ;
PUSH   ArrayCountPtr            ;

CALL   VDHLockMem               ; Call the function
                                ; Notice that it isn't necessary to pop the variable that was
                                ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
StartingLinAddr	DD	Starting address of the region in the user process that is to be locked
NumBytes	DD	Size, in bytes, of the region to lock
OptionFlag	DD	Used to set access options for the locked region. See below.
PagelistArrayPtr	DD	Pointer to array of VDHPageList_s structures. See below.
ArrayCountPtr	DD	Pointer to a variable where a count of the number of elements returned in the PagelistArrayPtr array is placed by VDHLockMem. If PagelistArrayPtr is set to VDHLM_NO_ADDR, ArrayCountPtr is ignored.

OptionFlag Used to set access options for the locked region. The VDHLM_READ and VDHLM_WRITE flags are for verification purposes and are not mixed with the VDHLM_RESIDENT flag. Either the VDHLM_READ, VDHLM_WRITE, or VDHLM_RESIDENT flag must be used. Notice that VDHLM_RESIDENT cannot be combined with other flags. Possible values are:

VDHLM_RESIDENT Lock physical pages.

VDHLM_READ Verify read access. VDHLM_READ is a verify lock, and is not mixed with other flags. Non-verify locks always make the region resident.

VDHLM_WRITE Verify write access. VDHLM_WRITE is a verify lock, and is not mixed with other flags. Non-verify locks always make the region resident.

VDHLM_CONTIGUOUS	Force physical pages contiguous; can be used only for 64KB or less.
VDHLM_NOBLOCK	Lock must not give up CPU
VDHLM_16M	Must reside below the 16MB physical memory address.

PagelistArrayPtr Pointer to an array of VDHPageList_s structures. VDHLockMem fills this array. Each VDHPageList_s structure describes a single physically contiguous sub-area of the physical memory that was locked. If PagelistArrayPtr is set to VDHLM_NO_ADDR, no array is returned. The area that PagelistArrayPtr points to must contain enough VDHPageList_s structures to handle a worst case of one structure per page plus one more structure, that is, the region does not begin on a page boundary.

Results

Success If the function is successful, it returns a lock handle. This is a global handle that can be used in any task context.

Failure If the function fails, it returns *0*. If the function fails, call VDHGetError to determine the nature of the problem. If NumBytes equals *0*, *a system halt occurs*.

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: There are no DOS Session termination implications for this function. There cannot be any memory locked on behalf of a particular DOS Session.

Notes: The caller must call VDHUnlockMem to remove this lock. All locks are long term. This function must be used prior to using any pointers passed to a virtual device driver from an OS/2 process (through DOSRequestVDD). This might also be useful to the callers of VDHCopyMem and VDHExchangeMem.

VDHMapPages, VDHReallocPages, and VDHFreePages block, if applied to pages that are locked. They unblock when the lock is released.

VDHMapPages

This function maps a linear address region in the V86 address space to:

- Part of a virtual memory object
- Specific physical addresses (for physical devices)
- Undefined memory (black holes)
- Invalid pages (the unmapped state, which is the default state for reserved pages).

VDHMapPages also changes the virtual address contents of regions in the V86 address space (below 11000H (1MB + 64KB)). Pages in this range can be made to address:

- Part of a memory object previously allocated using VDHAllocPages or VDHReallocPages
- Physical addresses for a device
- Black hole (undefined memory) addresses that are read or written but do not retain their values
- Invalid pages, which cause faults when accessed (see VDHInstallFaultHook).

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHMapPages:NEAR
pvdhmsSourceRegion     DD     ?       ; Source region definition
pvdhmtTargetRegion     DD     ?       ; Target region definition
flMappingFlag          DD     ?       ; Mapping options flag

PUSH   pvdhmsSourceRegion             ; Push the parameters
PUSH   pvdhmtTargetRegion             ;
PUSH   flMappingFlag                  ;

CALL   VDHMapPages                    ; Call the function.
                                      ; Notice that it isn't necessary to pop the variable that was
                                      ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
pvdhmsSourceRegion	DD	Source region definition. Pointer to VDHMAPSOURCE structure. See below.
pvdhmtTargetRegion	DD	Target region definition. Pointer to VDHMAPTARGET structure. See below.
flMappingFlag	DD	Mapping options flag. See below.

pvdhmsSourceRegion VDHMAPSOURCE is the source region definition for VDHMapPages with a data structure as follows:

```
VDHMapSource_s STRUC
    vdhms_laddr    DD   ?      ; Source linear address of source memory object.
                               ; Source address to be mapped.  Defines the start of
                               ; the source region for the mapping.  The region
                               ; is the same length as the target region.
    vdhms_hobj     DD   ?      ; Memory object handle set by the service to hold a
                               ; source handle for VDHMT_LINEAR mappings.
                               ; See the VDHMT_LINEAR description under MappingFlag.

VDHMapSource_s ENDS
```

pvdhmtTargetRegion VDHMAPTARGET is the target region definition for VDHMapPages with a data structure as follows:

```
VDHMapTarget_s STRUC
    vdhmt_laddr   DD   ?   ; Address in V86-space to be mapped
                           ; (0 <= vdhmt_laddr < 1MB+64KB)
    vdhmt_cpg     DD   ?   ; Count of pages to map
    vdhmt_hmap    DD   ?   ; Mapping handle.  Must be zero on the first call to
                           ; VDHMapPages for region.  Set by the service to hold a
                           ; handle used for remapping the region.  The handle is
                           ; reset each time it is used.  vdhmt_hmap must be the
                           ; value returned from the previous VDHMapPages call for
                           ; the region unless the pages are either already invalid
                           ; or else are being made invalid.  If either the old or
                           ; new mapping is invalid pages, vdhmt_hmap can be 0.
                           ; Making a region invalid restores pages in the region
                           ; to their reserved state, and sets vdhmt_hmap to 0.
VDHMapTarget_s ENDS
```

MappingFlag Mapping options flag. Possible values are:

VDHMT_INVALID Make target region pages invalid. vdhmt_hmap can be *0*. This service is faster when the handle is provided but virtual device drivers that do not store handles can map to invalid before remapping a region. Notice that pvdhmsSourceRegion is ignored.

VDHMT_LINEAR Map linear source region into target region. vdhms_laddr contains the linear address to map at the start of the target region. This can only be a page-granular address in a memory allocation that was obtained by using VDHAllocPages or VDHReallocPages. After a successful mapping, the target region aliases the corresponding region of the source.

vdhms_hobj must be *0* the first time a newly allocated or reallocated source object is mapped. On a successful mapping, vdhms_hobj is filled in. Using this handle on the next mapping that uses the same source object speeds up that mapping call. Any linear address (laddr) at the start of the target region can be mapped using the handle. The returned result must be stored after each mapping call. Virtual device drivers can avoid storing this handle and always use *0*, however, the call will be slower.

VDHMT_PHYSICAL Map physical source region into target region. vdhms_laddr contains the physical address to map at the start of the target region. vdhms_hobj is ignored.

VDHMT_BLACK_HOLE Map target region to black hole pages. pvdhmsSourceRegion is ignored.

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, it returns 0. VDHGetError should be called to determine the nature of the problem.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: Mapping succeeds only if it completely replaces a previous mapping, or if the region contains only invalid pages. To change the size of a region, it must first be made invalid. The entire target region must lie below 11000H (1MB+64KB) and must be contained in pages previously reserved by VDHReservePages. VDHMapPages and VDHAllocPages can be used on the same target region but care must be taken. Before switching from one service to the other, the pages affected must first be made invalid. If pages are left invalid, a page fault handler must be registered (through VDHInstallFaultHook) to handle page faults, in case the pages are accessed.

Both target and source addresses must be on page boundaries. If they are not, the page offset part of the address is ignored.

VDHOpen

This function opens a file or device and returns a handle. The handle can be used with VDHRead, VDHWrite, VDHSeek, VDHClose, and VDHDevIOCtl. The parameters and error return codes are the same as those found in the DosOpen API. Refer to the *OS/2 2.0 Control Program Programming Reference* for complete descriptions of each DosOpen API parameter.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHOpen:NEAR
FileName         DD    ?      ; Pointer to the name of the file or device to open
FileHandle       DD    ?      ; Where the system returns the file handle
ActionTaken      DD    ?      ; Where a description of the action taken is returned
FileSize         DD    ?      ; The new file's logical size (EOD) in bytes
FileAttribute    DD    ?      ; File attribute bits, used on file creation
OpenFlag         DD    ?      ; Specifies action taken, depending on whether file exists
OpenMode         DD    ?      ; Describes the mode of the open function
EABuf            DD    ?      ; Address of a VDH_EAOP structure or NULL

PUSH   FileName               ; Push the parameters
PUSH   FileHandle             ;
PUSH   ActionTaken            ;
PUSH   FileSize               ;
PUSH   FileAttribute          ;
PUSH   OpenFlag               ;
PUSH   OpenMode               ;
PUSH   EABuf                  ;

CALL   VDHOpen                ; Call the function.
                              ; Notice that it is not necessary to pop the variable pushed;
                              ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
FileName	DD	Pointer to the ASCIIZ string containing the name of the device or file to open.
FileHandle	DD	Where the system returns the file handle.
ActionTaken	DD	Where the system returns a description of the action taken as a result of this function call.
FileSize	DD	The new file's logical size (EOD) in bytes.
FileAttribute	DD	File attribute bits used on file creation.
OpenFLag	DD	Specifies the action taken depending on whether the file exists. Possible actions taken are: **VDHOPEN_FILE_EXISTED** File existed **VDHOPEN_FILE_CREATED** File created **VDHOPEN_FILE_TRUNCATED** File replaced.
OpenMode	DD	Describes the mode of the Open function.
EABuf	DD	Address of a VDH_EAOP structure, or NULL.

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, it returns 0. If the function fails, VDHGetError should be called to determine the nature of the problem. VDHGetError can return the following errors:

ERROR_DISK_FULL
ERROR_OPEN_FAILED
ERROR_FILE_NOT_FOUND
ERROR_INVALID_PARAMETER
ERROR_PATH_NOT_FOUND
ERROR_ACCESS_DENIED
ERROR_DRIVE_LOCKED
ERROR_NOT_DOS_DISK
ERROR_SHARING_BUFFER&US.EXCEEDED
ERROR_SHARING_VIOLATION
ERROR_INVALID_ACCESS
ERROR_CANNOT_MAKE
ERROR_TOO_MANY_OPEN_FILES
ERROR_NOT_ENOUGH_MEMORY.

If the function is called at other than DOS Session-task time, or if the pszFileName pointer is invalid, *a system halt occurs*.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: Any handles the virtual device driver opened for the terminating DOS Session must be closed with VDHClose.

Notes: Using VDHOpen does not interfere with the handle space available to the DOS Session with INT 21H, except that it does count against the system wide limit on the number of open file handles. These VDHOpen handles and the handles returned from INT 21H $Open cannot be used interchangeably.

VDHOpenPDD

This function gets the address of the routine in a physical device driver that the virtual device driver uses to communicate with the physical device driver. The physical device driver's device must have previously registered its name and entry point using the DevHlp service, RegisterPDD.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHOpenPDD:NEAR
DeviceName        DD   ?     ; Pointer to PDD device name to open
VDDEntryPoint     DQ   ?     ; Virtual device driver entry point for use by the
                             ; physical device driver in communicating with
                             ; the virtual device driver.

PUSH   DeviceName            ; Push the parameters
PUSH   VDDEntryPoint         ;

CALL   VDHOpenPDD            ; Call the function
                             ; Notice that it is not necessary to pop the variable pushed;
                             ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
DeviceName	DD	Pointer to the device name of the physical device driver to open.
VDDEntryPoint	DQ	Virtual device driver entry point for use by the physical device driver. Since physical device drivers are 16:16 modules, this is a 16:32 entry point. See below for details.

The interface for the VDDEntryPoint:

```
EntryPoint  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 12] - ulFunc (DD) - Function code
;                        All function codes are private to a PDD/VDD pair.
;           [ESP +  8] - ul1 (DD)    - Function-specific
;                        If a pointer, it is 16:16, and will generally
;                        be an input packet.
;           [ESP +  4] - ul2 (DD)    - Function-specific
;                        If a pointer, it is 16:16, and will generally
;                        be an output packet.
;    EXIT  SUCCESS  Returns a non-zero value in EAX
;          FAILURE  Returns 0 in EAX
```

Results

Success — If the function is successful, it returns a pointer to the physical device driver Inter-Device Communication (IDC) function. See "RegisterPDD" in the *OS/2 2.0 Physical Device Driver Reference* for a description of the physical device driver's IDC function.

Failure — If the function fails, it returns 0. VDHGetError should be called to determine the nature of the problem.

Remarks:

Context Issues: This function can be called only in the initialization context.

DOS Session Terminations: The virtual device driver must communicate with the physical device driver to ensure that any resources the physical device driver has for the terminating DOS Session are freed.

VDHOpenVDD

This function returns a handle to a virtual device driver that can be used with VDHRequestVDD to enable a virtual device driver to communicate with another virtual device driver. The name must be one that was previously registered by using VDHRegisterVDD.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHOpenVDD:NEAR
VDDName        DD    ?   ; Pointer to the name of the VDD to open

PUSH   VDDName           ; Push the parameter

CALL   VDHOpenVDD        ; Call the function
                         ; Notice that it is not necessary to pop the variable pushed;
                         ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
VDDName	DD	Pointer to a string containing the name of the virtual device driver to open.

Results

Success If the function is successful, it returns a handle to the virtual device driver.

Failure If the function fails, it returns 0. This can occur if:

- The virtual device driver referenced by VDDName is not found.
- The virtual device driver was found, but has not registered a worker function (the routine that gets control when this interface is called) for virtual device driver/virtual device driver communication.

If this function is called with invalid parameters or in the incorrect context, *a system halt occurs.*

Remarks:

Context Issues: This function can be called only in the DOS Session-task context, typically during DOS Session creation.

DOS Session Terminations: The virtual device driver must communicate with the opened virtual device driver to ensure that any resources for the terminating DOS Session are freed. In addition, the virtual device driver must communicate with any virtual device driver, or OS/2 clients, to ensure that any resources created due to the connection are freed.

Notes: The recommended way for two virtual device drivers to rendezvous with this function is for each to attempt to open the other at the first creation of a DOS Session. This allows the two virtual device drivers to work together regardless of the order in which they are initialized.

VDHOpenVIRQ

This function returns an IRQ handle for use with the other Virtual Programmable Interrupt Controller (VPIC) services and optionally sets handlers called when an End Of Interrupt (EOI) or Interrupt Return (IRET) is executed during a simulated interrupt.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHOpenVIRQ:NEAR
IRQNumber      DD     ?    ; Number of the IRQ to open
EOIHandler     DD     ?    ; Address of End-Of-Interrupt (EOI) handler
IRETHandler    DD     ?    ; Address of Interrupt Return (IRET) handler
Timeout        DD     ?    ; IRET timeout in milliseconds
OptionFlag     DD     ?    ; Indicates whether or not the virtual device driver will share an IRQ

PUSH   IRQNumber           ; Push the parameters
PUSH   EOIHandler          ;
PUSH   IRETHandler         ;
PUSH   Timeout             ;
PUSH   OptionFlag          ;

CALL   VDHOpenVIRQ         ; Call the function
                           ; Notice that it is not necessary to pop the variable pushed;
                           ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
IRQNumber	DD	Number of the IRQ to open.
EOIHandler	DD	Address of End-Of-Interrupt (EOI) handler. See below.
IRETHandler	DD	Address of Interrupt Return (IRET) handler. See below.
Timeout	DD	IRET timeout value in milliseconds. When the timeout expires, the virtual device driver's IRET handler is called as if the IRET had occurred. A value of −1 indicates that no time out occurs.
OptionFlag	DD	Option flag. See below.

EOIHandler Address of End-Of-Interrupt (EOI) handler. Linear address of the handler to be called when EOI is received by VPIC from a DOS Session for the IRQ. If this is not desired, put a *0* in this parameter. Notice that the IRQ is no longer *in service* (the ISR bit has been cleared) when the virtual device driver's EOI handler is called. The interface to the EOI handler is:

```
EOIHdlr  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 4] - pcrf - Pointer to client register frame
;    EXIT   None
;
; USES      EAX, ECX, EDX, FS, GS, Flags
; CONTEXT   DOS Session-task
```

IRETHandler Address of Interrupt Return (IRET) handler. Linear address of the handler to be called when the IRET in the DOS Session's interrupt code is executed for this IRQ. If this is not desired, set the parameter to *0*. The interface for the IRET routine is:

```
IRETHdlr  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 4] - pcrf - Pointer to client register frame
;    EXIT   None
;
; USES      EAX, ECX, EDX, FS, GS, Flags
; CONTEXT   DOS Session-task
```

OptionFlag Indicates whether the IRQ can be shared or not. Possible value:

VPIC_SHARE_IRQ Indicates that the virtual device driver is willing to share the IRQ with another virtual device driver. All virtual device drivers that virtualize the same IRQ must pass this flag. Whether a particular IRQ is shared or unshared is determined by the setting of OptionFlag by the first virtual device driver to open the IRQ.

Results

Success If the function is successful, it returns an IRQ handle.

Failure If the function fails, it returns 0. VDHGetError should be called to determine the nature of the problem. If IRQNumber, EOIHandler, IRETHandler, or OptionFlag are invalid, *a system halt occurs*.

Remarks:

Context Issues: This function can be called only in the initialization context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: The maximum number of virtual device drivers that can share an IRQ is *32*.

VDHPhysicalDisk

This function returns information about partitionable disks.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHPhysicalDisk:NEAR
Function      DD    ?      ; Type of partitionable disk information to obtain
DataPtr       DD    ?      ; Pointer to the return buffer
DataLen       DD    ?      ; Length of the return buffer
ParmPtr       DD    ?      ; Pointer to the user-supplied information
ParmLen       DD    ?      ; Length of the user-supplied information

PUSH   Function             ; Push the parameters
PUSH   DataPtr              ;
PUSH   DataLen              ;
PUSH   ParmPtr              ;
PUSH   ParmLen              ;

CALL   VDHPhysicalDisk      ; Call the function
                            ; Notice that it is not necessary to pop the variable pushed;
                            ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
Function	DD	Type of partitionable disks information to obtain. See below.
DataPtr	DD	Pointer to the return buffer
DataLen	DD	Length of the return buffer. See below.
ParmPtr	DD	Pointer to the user-supplied information.
ParmLen	DD	Length of the user-supplied information. See below.

Function Type of partitionable disks information to obtain. Possible values are:

VDHPHYD_GET_DISKS	Obtain the total number of partitionable disks
VDHPHYD_GET_HANDLE	Obtain a handle to use with the Category 9 IOCtls
VDHPHYD_RELEASE_HANDLE	Release a handle for a partitionable disk.

DataLen Length of the return buffer. The returned data for each function is described below (all lengths are in bytes):

VDHPHYD_GET_DISKS	Data Length = 2. Total number of partitionable disks in the system. 1-based.
VDHPHYD_GET_HANDLE	Data Length = 2. Handle for the specified partitionable disk for Category 9 IOCtls.
VDHPHYD_RELEASE_HANDLE	Data Length = 0. None, pointer must be *0*.

ParmLen Length of the user-supplied information (all lengths are in bytes):

VDHPHYD_GET_DISKS	Parm Length = 0. None, must be set to *0*. 1-based.
VDHPHYD_GET_HANDLE	Parm Length = string length. ASCIIZ string that specifies the partitionable disk.
VDHPHYD_RELEASE_HANDLE	Parm Length = 2. Handle obtained from VDHPHYD_GET_HANDLE.

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, it returns 0. VDHGetError should be called to determine the nature of the problem. VDHGetError can return the following error return codes:

ERROR_ACCESS_DENIED
ERROR_INVALID_FUNCTION
ERROR_INVALID_HANDLE
ERROR_INVALID_PARAMETER
ERROR_LOCK_VIOLATION

If VDHPhysicalDisk is called in any context except DOS Session task time, *a system halt occurs*.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: The handle obtained with Function Value=2 is released with Function Value=3. The ASCIIZ string used to specify the partitionable disk must be of the following format:

```
number:<null byte>

Where:
      number       Specifies the partitionable disk (1-based) number in ASCII
      :            Must be present
      <null byte>  Specifies the byte of 0 for the ASCIIZ string
```

Notes: If the pointers passed by this function are allocated from the stack, then the SSToDS macro must be used to make the pointer, DS, relative. Addresses (pointers) inside of device-specific Data and Parameter Packets are not translated.

VDHPopInt

This function reverses the effects of VDHPushInt. It removes the IRET frame from the client's stack and restores the client's CS:IP to what was in the IRET frame.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHPopInt:NEAR

CALL   VDHPopInt          ; Call the function
                          ; Notice that it is not necessary to pop the variable pushed;
                          ; the Virtual DevHlp function does this.
```

Parameters: None.

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, it returns 0 if the client is in protect mode and the stack is invalid or a stack overflow would occur. A stack exception will be simulated to the client.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: This function always returns successful when the DOS Session is in V86 mode because no data accesses below 1MB can cause a fault.

VDHPopRegs

This function reverses the effect of VDHPushRegs by popping the specified client registers into the Client Register Frame (CRF) from the client's stack.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHPopRegs:NEAR
RegFlag     DD     ?       ; Indicates which client registers to pop

PUSH   RegFlag             ; Push the parameter

CALL   VDHPopRegs          ; Call the function
                           ; Notice that it is not necessary to pop the variable pushed;
                           ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
RegFlag	DD	Flag indicating which client registers to pop. See below.

RegFlag Flag indicating which client registers to pop. These flags can be ORed together to indicate more than one register. Possible values are:

VDHREG_AX	Pop the AX register
VDHREG_BX	Pop the BX register
VDHREG_CX	Pop the CX register
VDHREG_DX	Pop the DX register
VDHREG_SI	Pop the SI register
VDHREG_DI	Pop the DI register
VDHREG_BP	Pop the BP register
VDHREG_SP	Pop the SP register
VDHREG_DS	Pop the DS register
VDHREG_ES	Pop the ES register
VDHREG_SS	Pop the SS register
VDHREG_FLAG	Pop the Flags register
VDHREG_ALL	Pop all the registers
VDHREG_GENERAL	Pop all the registers except SS and SP.

Results

Success If the function was successful, it returns a non-zero value.

Failure If this function fails, it returns 0 if the client is in protect mode and the stack is invalid or a stack underflow would occur. A stack exception is simulated to the client. If RegFlag is invalid, *a system halt occurs*.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: The caller should be careful to pass the same RegFlag value to VDHPopRegs that was used for the corresponding VDHPushRegs call. If the caller does not do this, the V86 stack can become corrupted. This function always returns successful when RegFlag is valid and the DOS Session is in V86 mode because no data accesses below 1MB can cause a fault.

VDHPopStack

This function pops the data off the client's stack.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHPopStack:NEAR
NumBytes    DD     ?     ; Number of bytes to pop off the stack
DataPtr     DD     ?     ; Pointer to the data to pop off the stack

PUSH   NumBytes          ; Push the parameter
PUSH   DataPtr           ;

CALL   VDHPopStack       ; Call the function
                         ; Notice that it is not necessary to pop the variable pushed;
                         ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
NumBytes	DD	Number of bytes to pop off the client's stack. Must be an even number.
DataPtr	DD	Pointer to the data to pop off the client's stack.

Results

Success If the function was successful, it returns a non-zero value.

Failure If this function fails, it returns 0 if a fault would occur when accessing the user stack. If NumBytes is odd, *a system halt occurs*.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: This is a very low-level service. Appropriate care should be exercised in its use to ensure that the client's stack is not corrupted. This service handles stack wraparound, and always returns successful when the DOS Session is in V86-mode because no data accesses below 1MB can cause a fault.

VDHPopup

This function displays a message according to the Message ID, and gets a response from the user.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHPopup:NEAR
ParmTable       DD     ?     ; Pointer to a table of substitution strings
StrCount        DD     ?     ; Number of substitution strings
MsgNumber       DD     ?     ; Message number
RespPtr         DD     ?     ; Pointer to a DD to receive the response
RespAllowed     DD     ?     ; Bit flag describing the allowed responses
Reserved        DD     ?     ; Reserved.  Must be set to NULL.

PUSH   ParmTable             ; Push the parameters
PUSH   StrCount              ;
PUSH   MsgNumber             ;
PUSH   RespPtr               ;
PUSH   RespAllowed           ;
PUSH   Reserved              ;

CALL   VDHPopup              ; Call the function.
                             ; Notice that it is not necessary to pop the variable pushed;
                             ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
ParmTable	DD	Pointer to a table of substitution strings.
StrCount	DD	Number of substitution strings.
MsgNumber	DD	Message number.
RespPtr	DD	Pointer to a DD to receive the returned response, filled on exit.
RespAllowed	DD	Bit field describing the allowed responses. The allowed values are below (any combination of these flags can be specified): **VDHP_FAIL** (0001H) **VDHP_RETRY** (0004H) **VDHP_IGNORE** (0008H) **VDHP_TERMINATE_SESSION** (0002H).
Reserved	DD	Reserved. Must be set to NULL.

Results

Success — If the function is successful, it returns a non-zero value and the variable pointed to by RespPtr is filled in with actual response selected by the user.

Failure — If the function fails, it returns 0. VDHGetError should be called to determine the nature of the problem.

Remarks:

Context Issues: This function can be called in the initialization context or the DOS Session-task context. If it is called at initialization time, the message is displayed as a string on the console as if a DosWrite to standard output (stdout) had been made. No prompt is displayed, and no user response is returned. ResponsePtr is filled in with *0*.

DOS Session Terminations: A DOS Session cannot be terminated if it is waiting in this call for a user response.

Notes: This service is intended for virtual device drivers, which must inform the user of some extraordinary circumstance. For example, the virtual COM device driver (VCOM) issues this call to inform the user that a COM port is busy, and gives the user a choice of how to handle the situation.

The input value of ResponsePtr is not used. *Retry* is the default action chosen for hard errors. If retry is not allowed, *End the Program* is chosen as the default.

VDHPostEventSem

This function posts an event semaphore. All the threads blocked on this semaphore will wake up.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHPostEventSem:NEAR
EventSemHandle     DD    ?     ; Handle to an event semaphore

PUSH   EventSemHandle          ; Push the parameter

CALL   VDHPostEventSem         ; Call the function
                               ; Notice that it is not necessary to pop the variable pushed;
                               ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
EventSemHandle	DD	Handle of the event semaphore to post.

Results

Success If the function is successful, it will return nothing.

Failure Posting a semaphore that is invalid or is already posted *causes a system halt*.

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHPrintClose

This function is called by other virtual device drivers to flush and close a DOS Session's open printers that have been opened for INT 17H printing. A DOS Session can have more than one printer open. This service flushes and closes all of them.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHPrintClose:NEAR
VDMHandle     DD     ?        ; Handle to a DOS Session

PUSH   VDMHandle              ; Push the parameter

CALL   VDHPrintClose          ; Call the function
                              ; Notice that it is not necessary to pop the variable pushed;
                              ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
VDMHandle	DD	DOS Session handle.

Results: None.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: The system will flush and close any open printers when the DOS Session is terminated.

Notes: If a printer has been left open because of direct access to the parallel ports, it will also be closed.

VDHPushFarCall

This function simulates a far call to V86 code as if a DOS program had executed a Far Call instruction.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHPushFarCall:NEAR
SegOffAddr     DQ     ?   ; V86-mode SEG:OFF address of V86-mode code to call

PUSH   SegOffAddr         ; Push the parameter

CALL   VDHPushFarCall     ; Call the function
                          ; Notice that it is not necessary to pop the variable pushed;
                          ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
SegOffAddr	DQ	The V86-mode segment:offset address of the V86-mode code to call. A pseudo V86 pointer.

Results

Success If the function was successful, it returns a non-zero value.

Failure If the function fails, it returns 0 if the client is in protect mode and the stack is invalid or a stack underflow would occur. A stack exception is simulated to the client.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: This function does not take effect until the operating system returns to V86 mode. In most cases, VDHArmReturnHook is called after this service. This allows the virtual device driver to regain control when the called V86 code executes its Far Return (RETF).

This function always returns successful when the DOS Session is in V86 mode because no data accesses below 1MB can cause a fault.

VDHPushInt

This function is used to change the DOS Session's control flow to the V86-interrupt handler when an interrupt is simulated. It returns to the DOS Session-interrupt code service. This function also simulates an interrupt to a specified V86-interrupt vector as if a DOS program has executed an INT *n* instruction. Any virtual device driver hooked on this interrupt with VDHInstallIntHook will get control as expected. The VDHArmReturnHook service can be used after this function is called to regain control when the V86 code executes the Interrupt Return (IRET).

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHPushInt:NEAR
Vector     DD     ?        ; Number of the interrupt vector to push

PUSH   Vector              ; Push the parameter

CALL   VDHPushInt          ; Call the function
                           ; Notice that it is not necessary to pop the variable pushed;
                           ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
Vector	DD	Number of the interrupt vector to push (*0 – 255*).

Results

Success If the function was successful, it returns a non-zero value.

Failure If the function fails, it returns 0 if the client is in protect mode and the stack is invalid or a stack underflow would occur. A stack exception will be simulated to the client. If Vector is invalid, *a system halt occurs.*

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: This function does not take effect until the operating system returns to V86 mode. This function always returns successful when the DOS Session is in V86 mode because no data accesses below 1MB can cause a fault.

VDHPushRegs

This function pushes the specified client register from the Client Register Frame (CRF) to the client's stack. This service is usually used in conjunction with VDHPushFarCall to preserve user registers.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHPushRegs:NEAR
RegFlag     DD     ?    ; Flag indicating which client registers to push

PUSH   RegFlag          ; Push the parameter

CALL   VDHPushRegs      ; Call the function
                        ; Notice that it is not necessary to pop the variable pushed;
                        ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
RegFlag	DD	Flag indicating which client registers to push. See below.

RegFlag Flag indicating which client registers to push. These flags can be ORed together to indicate more than one register. Possible values are:

VDHREG_AX	Push the AX register
VDHREG_BX	Push the BX register
VDHREG_CX	Push the CX register
VDHREG_DX	Push the DX register
VDHREG_SI	Push the SI register
VDHREG_DI	Push the DI register
VDHREG_BP	Push the BP register
VDHREG_SP	Push the SP register
VDHREG_DS	Push the DS register
VDHREG_ES	Push the ES register
VDHREG_SS	Push the SS register
VDHREG_FLAG	Push the Flags register
VDHREG_ALL	Push all the registers
VDHREG_GENERAL	Push all the registers except SS and SP.

Results

Success If the function was successful, it returns a non-zero value.

Failure If the function fails, it returns 0 if the client is in protect mode and the stack is invalid or a stack underflow would occur. A stack exception will be simulated to the client. If RegFlag is invalid, *a system halt occurs*.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: Because the registers are saved on the DOS Session stack (as opposed to a virtual device driver data structure), the virtual device driver is in no danger of losing track of data if the DOS Session does not return to the virtual device driver. This service is used sparingly as it consumes DOS Session stack space and most DOS applications cannot tolerate excessive stack usage beyond their own needs.

This function always returns successful when RegFlag is valid and the DOS Session is in V86 mode because no data accessed below 1MB can cause a fault.

VDHPushStack

This function pushes data onto the client's stack.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHPushStack:NEAR
NumBytes     DD     ?     ; Number of bytes to push on the stack
DataPtr      DD     ?     ; Pointer to the data to push on the stack

PUSH   NumBytes           ; Push the parameters
PUSH   DataPtr            ;

CALL   VDHPushStack       ; Call the function
                          ; Notice that it is not necessary to pop the variable pushed;
                          ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
NumBytes	DD	Number of bytes to push onto the client's stack. Must be an even number.
DataPtr	DD	A pointer to the data to be pushed.

Results

Success If the function was successful, it returns a non-zero value.

Failure If the function fails, it returns 0 if the client is in protect mode and the stack is invalid or a stack underflow would occur. A stack exception will be simulated to the client. If NumBytes is odd, *a system halt occurs.*

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: This is a very low-level service. Appropriate care should be exercised in its use to ensure that the client's stack is not corrupted. This service handles stack wraparound, and always returns successful when the DOS Session is in V86-mode because no data accesses below 1MB can cause a fault.

VDHPutSysValue

This function sets a system value.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHPutSysValue:NEAR
Index     DD     ?       ; Index of the system variable to set
Value     DD     ?       ; Value to set the system variable to

PUSH   Index             ; Push the parameters
PUSH   Value             ;

CALL   VDHPutSysValue    ; Call the function
                         ; Notice that it is not necessary to pop the variable pushed;
                         ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
Index	DD	Index of the system value to set. System value indexes are defined in VDMM.INC and listed in VDHQuerySysValue on page 5-113.
Value	DD	New value for the system variable.

Results

Success If the function was successful, it returns nothing.

Failure If Index is invalid, *a system halt occurs*.

Remarks:

Context Issues: This function can be called in the initialization or DOS Session-task (DOS Session creation) context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: This service is intended for system virtual device drivers to export system constants such as ROM memory size. The behavior of the system is undefined, if this service is used for any other purpose.

VDHQueryA20

This function returns the A20 state of the current DOS Session.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHQueryA20:NEAR

CALL   VDHQueryA20        ; Call the function
                          ; Notice that it is not necessary to pop the variable pushed;
                          ; the Virtual DevHlp function does this.
```

Parameters: None.

Results

Success If A20 is enabled (and wrapping is *off*), this function returns a non-zero value. If A20 is disabled (and wrapping is *on*), this function returns 0.

Failure If the function fails, it returns nothing.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHQueryFreePages

This function returns the total amount of free virtual memory in bytes. This is the amount of memory that could be allocated successfully.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHQueryFreePages:NEAR

CALL   VDHQueryFreePages     ; Call the function
                             ; Notice that it is not necessary to pop the variable pushed;
                             ; the Virtual DevHlp function does this.
```

Parameters: None.

Results

Success If the function is successful, it returns the number of bytes of free virtual memory.

Failure If the function fails, it returns nothing.

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: This is a system-wide value that is not specific to the virtual address space of the DOS Session.

VDHQueryHookData

This function returns a pointer to the reference data created during the call to VDHAllocHook.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHQueryHookData:NEAR
HookHandle     DD     ?     ; Hook handle from VDHAllocHook

PUSH   HookHandle           ; Push the parameter

CALL   VDHQueryHookData     ; Call the function
                            ; Notice that it is not necessary to pop the variable pushed;
                            ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
HookHandle	DD	Hook handle (from VDHAllocHook) for the hook to query.

Results

Success If the function is successful, it returns a pointer to the reference data block.

Failure If HookHandle is invalid or if there is no reference data (that is, the RefDataSize parameter for VDHAllocHook is *0*), *a system halt occurs.*

Remarks:

Context Issues: This function can be called in the initialization, task, or interrupt context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHQueryLin

This function converts a 16:16 (Far16) address to a 0:32 address. The 16:16 address can be a LDT-based or GDT-based address. The function is also callable on stack-based addresses.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHQueryLin:NEAR
Far16Addr     DD     ?     ; 16:16 address

PUSH   Far16Addr           ; Push the parameter

CALL   VDHQueryLin         ; Call the function
                           ; Notice that it is not necessary to pop the variable pushed;
                           ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
Far16Addr	DD	The 16:16 address to be converted to a 0:32 address.

Results

Success If the function is successful, it returns the 0:32 address that corresponds to Far16Addr.

Failure If Far16Addr is an invalid selector, *a system halt occurs.*

Remarks:

Context Issues: This function can be called in the initialization, task, or interrupt context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: This service does minimal checking on Far16Addr. The caller should make sure that Far16Addr is not invalid. If an invalid address is passed, this service can return an invalid address.

VDHQueryKeyShift

This function is called by the virtual Mouse device driver to query the keyboard shift state of a DOS Session.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHQueryKeyShift:NEAR
VDMHandle     DD     ?     ; DOS Session handle

PUSH   VDMHandle           ; Push the parameter

CALL   VDHQueryKeyShift    ; Call the function
                           ; Notice that it is not necessary to pop the variable pushed;
                           ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
VDMHandle	DD	DOS Session handle.

Results: None.

Remarks:

Context Issues: This function can be called only in the interrupt context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHQueryProperty

This function returns the current value of the DOS Setting specified.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHQueryProperty:NEAR
PropertyName    DD    ?    ; Pointer to a property name

PUSH   PropertyName        ; Push the parameter

CALL   VDHQueryProperty    ; Call the function.
                           ; Notice that it is not necessary to pop the variable pushed;
                           ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
PropertyName	DD	Pointer to an ASCIIZ string containing the property name for which information is being sought. Maximum length is 40 characters, including the terminating NULL.

Results

Success The format of the return value for successful depends on the type of the property being queried:

VDMP_BOOL The return value is a BOOL (Boolean). A *0* value represents FALSE and a non-zero value represents TRUE.

VDMP_INT The return value is a DD. It is guaranteed to be valid with respect to the bounding information specified in VDHRegisterVDD. Only the low half (DW) of the DD is significant. The high half is always *0*.

VDMP_ENUM The return value is a pointer to a NULL-terminated string. The string is guaranteed to be one of those specified in VDHRegisterVDD.

VDMP_STRING The return value is a pointer to a NULL-terminated string. The string is guaranteed to be at most as long as the limit specified in VDHRegisterVDD.

VDMP_MLSTR The return value is a pointer to a NULL-terminated string. The string is guaranteed to be at most as long as the limit specified in VDHRegisterVDD.

Note: For property types VDMP_ENUM, VDMP_STRING, and VDMP_MLSTR, a pointer to the current value is returned to the virtual device driver, thereby avoiding a condition where an OS/2 process might be trying to change the same value. The virtual device driver calls VDHFreeMem to free the string after the virtual device driver is finished with it.

Failure If the function fails, it returns NULL for property types VDMP_ENUM, VDMP_STRING, and VDMP_MLSTR if there is insufficient memory to create a copy of the string value. In this case, the virtual device driver uses the default property value. NULL is also returned if pszName is invalid or is not a registered virtual device driver name.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: All memory allocated by a virtual device driver for use by the terminating DOS Session is freed by using VDHFreeMem.

Notes: A virtual device driver can assume that the property value is valid. The system validates all types except VDMP_STRING and VDMP_MLSTR by using the validation values passed on the call to VDHRegisterProperty. The string types are validated by calling the virtual device driver function registered through VDHRegisterProperty.

VDHQuerySel

This function returns the selector part of a 16:16 far pointer from a flat 0:32 address. The selector is in the Global Descriptor Table (GDT).

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHQuerySel:NEAR
VirtAddress     DD     ?  ; 0:32 virtual address

PUSH   VirtAddress        ; Push the parameter

CALL   VDHQuerySel        ; Call the function
                          ; Notice that it is not necessary to pop the variable pushed;
                          ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
VirtAddress	DD	A 0:32 virtual address.

Results

Success If the function is successful, it returns a non-zero selector.

Failure If the specified linear address is not in a virtual device driver data object, *a system halt occurs*.

Remarks:

Context Issues: This function can be called in the initialization and task context:

- During initialization time; query is allowed only on initialization and global data.
- Global, instance, and stack data are allowed during task time (typically DOS Session creation time), but initialization data is not.

DOS Session Terminations: Selectors for virtual device driver instance data are destroyed when the DOS Session terminates. The virtual device driver must ensure that any physical device driver that was given this kind of selector is finished with it.

Notes: This function works only on 0:32 addresses in virtual device driver data objects and stack. Each virtual device driver data object is limited to a maximum of 64KB in size so that the following formula can be used. A virtual device driver that needs more than 64KB of a particular class (INIT, global, or instance) must use multiple objects.

Notice that the selectors for virtual device driver data are created to map starting at a 64KB linear address boundary. As a result, VDHQuerySel needs to be called only once, and the returned selector is valid for any memory in the particular object that contains the passed address. This is because the DOS Session Manager does not allow a virtual device driver data object to span a 64KB linear address boundary.

For virtual device driver instance data, this function returns a different GDT selector for each DOS Session, and it can be called only in the context of a DOS Session.

VDHQuerySem

This function queries the state of an event or mutex semaphore.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHQuerySem:NEAR
SemHandle       DD     ?       ; Handle to an event or mutex semaphore
SemStatePtr     DD     ?       ; Pointer to a VDHSEMSTATE data structure

PUSH   SemHandle               ; Push the parameters
PUSH   SemStatePtr             ;

CALL   VDHQuerySem             ; Call the function
                               ; Notice that it is not necessary to pop the variable pushed;
                               ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
SemHandle	DD	Semaphore handle
SemStatePtr	DD	A pointer to the data area to be filled with the state of the semaphore (see structure below).

SemStatePtr points to a variable defined in VDMM.INC and structured as follows:

```
VDHSemState_s STRUC
        vss_SemType     DB   ?   ; VDH_EVENTSEM/VDH_MUTEXSEM
        vss_fOwned      DB   ?   ; 0 = Not Owned; 1 = Owned
        vss_fWaiter     DW   ?   ; 0 = No one waiting; 1 = Waiting
        vss_cRequest    DW   ?   ; Request count in mutex case
        vss_tid         DW   ?   ; TID of the owner, if owned
VDHSemState_s ENDS
```

Results

Success If the function is successful, it returns nothing and the area pointed to by SemStatePtr is filled with the state of the semaphore in SemHandle.

Failure If SemHandle is invalid, *a system halt occurs*.

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: There are no DOS Session termination implications for state checking services.

Notes: The validity of SemStatePtr is not checked. The calling program must pass the address of a valid data area.

VDHQuerySysValue

This function queries a system value.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHQuerySysValue:NEAR
VDMHandle     DD     ?        ; DOS Session handle
Index         DD     ?        ; Index of the system variable

PUSH   VDMHandle              ; Push the parameters
PUSH   Index                  ;

CALL   VDHQuerySysValue       ; Call the function
                              ; Notice that it is not necessary to pop the variable pushed;
                              ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
VDMHandle	DD	Handle of the DOS Session to query. A value of *0* indicates the current DOS Session.
Index	DD	Index of the system variable. See below for values.

VDHQuerySysValue Index values (as defined in VDMM.INC) and descriptions:

Global Values	Ordinal	Type	Units	Range
VDHGSV_DAY	0	DD	days	1 < = x < = 31
VDHGSV_MONTH	1	DD	months	1 < = x < = 12
VDHGSV_YEAR	2	DD	years	1980 < = x < = MAXULONG
VDHGSV_DAYOFWEEK	3	DD	days	0 < = x < = 6
VDHGSV_HOUR	4	DD	hours	0 < = x < 24
VDHGSV_MINUTE	5	DD	minutes	0 < = x < 60
VDHGSV_SECOND	6	DD	seconds	0 < = x < 60
VDHGSV_HUNDREDTH	7	DD	1/100s	0 < = x < 100
VDHGSV_SECONDS1970	8	DD	seconds	0 < = x < = MAXULONG
VDHGSV_TIMEZONE	9	DD	minutes	0 < = x < = MAXULONG
VDHGSV_MSECSBOOT	10	DD	milliseconds	0 < = x < = MAXULONG
VDHGSV_TIMERINTERVAL	11	DD	milliseconds	0 < = x < = 1000
VDHGSV_DYNVARIATION	12	DD		TRUE (non-zero)/FALSE (0)
VDHGSV_MAXWAIT	13	DD	seconds	0 < = x < = MAXULONG
VDHGSV_MINTIMESLICE	14	DD	milliseconds	0 < = x < = MAXULONG
VDHGSV_MAXTIMESLICE	15	DD	milliseconds	0 < = x < = MAXULONG
VDHGSV_YIELD	16	DD		TRUE (non-zero)/FALSE (0)
VDHGSV_TCYIELD	17	DD		TRUE (non-zero)/FALSE (0)
VDHGSV_VERMAJOR	18	DD		0 < = x < = MAXULONG
VDHGSV_VERMINOR	19	DD		0 < = x < = MAXULONG
VDHGSV_VERREVISION	20	DB		A < = x < = Z
VDHGSV_MACHINETYPE	21	DD		MACHINE_TYPE_*
VDHGSV_BLACKHOLEADDR	22	DD	bytes	0 < = x < = MAXULONG
VDHGSV_BLACKHOLESIZE	23	DD	bytes	0 < = x < = MAXULONG
VDHGSV_FGNDSESSIONID	24	DD		0 < = x < MAXSESSIONS
VDHGSV_ARPLADDR	29	DD		
VDHGSV_MACHINEINFO	30	DD		Pointer to System Configuration table
VDHGSV_PPOSREGS	31	DD		Pointer to POS Regs structure
VDHGSV_PICMASK	32	DD		Original PIC mask values.

Local Values	Ordinal	Type	Units	Range
VDHLSV_HVDM	4096	DD		
VDHLSV_PID	4097	DW		
VDHLSV_PCRF	4098	DD		
VDHLSV_SESSIONID	4099	DD		N < = x < MAXSESSIONS
VDHLSV_FOREGROUND	4100	DD		TRUE (non-zero)/FALSE (0)
VDHLSV_RMSIZE	4101	DD	KB	0 < x < = 640
VDHLSV_CODEPAGEID	4102	DD		See DosGetCP
VDHLSV_PRIORITYCLASS	4103	DD		See VDHSetPriority
VDHLSV_PRIORITYLEVEL	4104	DD		See VDHSetPriority.
VDHLSV_VPICBASE	4105	DD		

Results

Success If the function is successful, it returns the value of the system variable. If the value is *0* and VDHGetError *returns* 0, then *0* is the correct value.

Failure If VDMHandle or Index is invalid, *a system halt occurs.*

Remarks:

Context Issues: This function can be called in any context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: Systems values are of two classes, global and per-DOS Session. These classes are distinguished by the prefix of the index name. Variables with prefixes of *VDHGSV_* are global. Variables with a *VDHLSV_* prefix are per-DOS Session.

VDHQueryVIRQ

This function returns information about the virtual mask, interrupt request flag, interrupt in-service flag, and pending virtual device driver Interrupt Return (IRET) handlers for the specified IRQ, device, or DOS Session.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHQueryVIRQ:NEAR
VDMHandle     DD   ?     ; DOS Session handle
IRQHandle     DD   ?     ; IRQ handle from VDHOpenVIRQ

PUSH   VDMHandle         ; Push the parameters
PUSH   IRQHandle         ;

CALL   VDHQueryVIRQ      ; Call the function
                         ; Notice that it is not necessary to pop the variable pushed;
                         ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
VDMHandle	DD	DOS Session handle. A value of *0* indicates the current DOS Session.
IRQHandle	DD	IRQ handle from VDHOpenVIRQ.

Results

Success If the function is successful, it returns the IRQ status as a DD of flag bits:

VPICQ_REQUEST_PENDING	Request is pending for the queried IRQ
VPICQ_IN_SERVICE	Queried IRQ is in service
VPICQ_VIRT_MASK	Mask is turned *on* for the queried IRQ
VPICQ_STAT_IRET_PENDING	IRET is pending for the queried IRQ.

Failure If VDMHandle or IRQHandle is invalid, *a system halt occurs.*

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHRaiseException

This function is used to raise an exception, which is reflected to a DOS Session as if the exception were caused by the hardware.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHRaiseException:NEAR
Exception      DD     ?
ErrorCode      DD     ?
ExtraCode      DD     ?

PUSH   Exception           ; Push the parameters
PUSH   ErrorCode           ;
PUSH   ExtraCode           ;

CALL   VDHRaiseException   ; Call the function
                           ; Notice that it is not necessary to pop the variable pushed;
                           ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
Exception	DD	An exception number as might be caused by the 80386 microprocessor
ErrorCode	DD	Error code for the exception as defined for the 80386 microprocessor
ExtraCode	DD	Extra error code for DPMI 1.0 page faults.

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, it returns 0.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHRead

This function reads bytes from a file or device previously opened by VDHOpen.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHRead:NEAR
FileHandle        DW    ?   ; Handle to the file/device to read from
ReadBufferPtr     DD    ?   ; Pointer to location to store the bytes read
NumBytes          DD    ?   ; Number of bytes to read

PUSH   FileHandle           ; Push the parameters
PUSH   ReadBufferPtr        ;
PUSH   NumBytes             ;

CALL   VDHRead              ; Call the function.
                            ; Notice that it is not necessary to pop the variable pushed;
                            ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
FileHandle	DW	Handle to the file or device to read
ReadBufferPtr	DD	Address of the buffer to store the bytes read
NumBytes	DD	Number of bytes to read.

Results

Success If the function is successful, it returns a count of the number of bytes read; this can be equal to *0*.

Failure If the function fails, it returns 0FFFFFFFFH. VDHGetError should be called to determine the nature of the problem. If FileHandle is invalid or this function is called in any context except DOS Session-task, *a system halt occurs.*

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHReadUBuf

This function is used to read from protect-mode address space as if the read were done at Ring 3. This means any other faults are trapped. All faults are passed on to the DOS Session application handler through VDHRaiseException.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHReadUBuf:NEAR
DestBuffer        DD     ?
ByteCount         DD     ?
Selector          DD     ?
OffsetPointer     DD     ?
Flag              DD     ?

PUSH   DestBuffer           ; Push the parameters
PUSH   ByteCount            ;
PUSH   Selector             ;
PUSH   OffsetPointer        ;
PUSH   Flags                ;

CALL   VDHReadUBuf          ; Call the function
                            ; Notice that it is not necessary to pop the variable pushed;
                            ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
DestBuffer	DD	Destination buffer to copy to
ByteCount	DD	Count of bytes
Selector	DD	Application selector
OffsetPointer	DD	Address of the variable containing the offset for the start of the read
Flag	DD	Flag containing checking controls. See below.

Flag Flag containing checking controls. Possible values are:

VPM_PROT_READ	Check for read
VPM_PROT_WRITE	Check for write
VPM_FAULT_IF_SU_SET	Fault, if supervisor pages
VPM_FAULT_IF_RO	Fault, if the descriptor is read only
VPM_SEL_PRESENT	Caller knows descriptor is present
VPM_SEL_WRITEABLE	Caller knows descriptor is writable
VPM_SEL_IS_SS	Selector is client's stack
VPM_XCPTRET_ALT	After exception, return to alternate mode. For example, if the client was in protect mode when the service was called, return in V86 mode after the exception is handled.

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, it returns 0 if a bad address reference is causing the fault. In this case, OffsetPointer is updated with the address of the fault. For selector faults, OffsetPointer is unchanged.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: If the routine fails, the caller must clean up and exit so that the exception can be simulated to the user. Supervisor pages can be read without faults to avoid the overhead of checking.

VDHReallocPages

This function expands or shrinks a memory object.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHReallocPages:NEAR
ObjectAddress        DD    ?      ; Address of the memory object to resize
NumPages             DD    ?      ; New size of the object (in 4KB pages)
Reserved             DD    ?      ; Must be zero.

PUSH   ObjectAddress              ; Push the parameters
PUSH   NumPages                   ;
PUSH   Reserved                   ;

CALL   VDHReallocPages            ; Call the function
                                  ; Notice that it is not necessary to pop the variable pushed;
                                  ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
ObjectAddress	DD	Address of the memory object to reallocate/resize
NumPages	DD	The new size of the memory object (in 4KB pages)
Reserved	DD	Must be set to zero.

Results

Success If the function is successful, it returns that address of the reallocated memory block.

Failure If the function fails, it returns *0*. VDHGetError should be called to determine the nature of the problem. If the memory object at ObjectAddress was not allocated by VDHAllocPages or VDHReallocPages, or if the NumPages or Reserved parameters are invalid, *a system halt occurs*.

Remarks:

Context Issues: This function can be called in the initialization or task context.

DOS Session Terminations: Any allocations that are not in the terminating DOS Sessions private area must be released by using VDHFreePages.

Notes: When an allocation made with VDHAllocPages is shrunk by VDHReallocPages, the linear range between the end of the allocation and the original end of the allocation remains available for object growth without movement. If the linear range encompassed by the new size was reserved (with VDHReservePages), reallocation occurs without movement. Reallocation also succeeds without movement, if the object was larger than the desired size since it last moved. Regardless of VDHReallocPages activity, all pages in the allocation retain the same properties, that is, fixed, system, and physical.

VDHRegisterDMAChannel

This function is used by virtual device drivers to register with the virtual DMA device driver (VDMA) to get the ownership of a DMA channel.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHRegisterDMAChannel:NEAR
DMAChannel          DD    ?       ; DMA channel
DMAHandlerFunc      DD    ?       ; The virtual device drivers DMA-handling routine

PUSH   DMAChannel                 ; Push the parameters
PUSH   DMAHandlerFunc             ;

CALL   VDHRegisterDMAChannel      ; Call the function.
                                  ; Notice that it is not necessary to pop the variable pushed;
                                  ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
DMAChannel	DD	DMA channel
DMAHandlerFunc	DD	The virtual device drivers DMA-handler function.

The interface for the DMA-handler routine is as follows:

```
DMAHandler  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 8] - hvdm   - DOS Session requesting DMA
;           [ESP + 4] - iEvent - VDD_DMA_MASKOFF (Start DMA, Event)
;                               VDD_DMA_MASK    (Stop  DMA, Event)
;    EXIT   SUCCESS - Returns non-0 in EAX
;                     If VDD_DMA_MASKOFF,
;                        the virtual DMA device driver will program the DMA
;                     If VDD_DMA_MASK, don't care
;           FAILURE - Returns 0 in EAX
```

Results

Success If the function is successful, it returns *1*.

Failure If the function fails, it returns *0*. VDHGetError should be called to determine the nature of the problem.

Remarks:

Context Issues: This function can be called only in the initialization context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: In the case of VDD_DMA_MASKOFF, the virtual device driver gets the hardware ownership and unhooks all the ports. If hardware ownership is not available immediately, it blocks the DOS Session. On VDD_DMA_MASK, the virtual device driver returns the ownership and hooks back the ports.

If the DOS application programs the hardware handled by a virtual device driver first and then programs the DMA, this virtual device driver receives ownership of hardware, unhooks all ports on first port access,

and arms a timer for a safe period. This is a preventative measure to protect the system from errant I/O to DMA ports. If the timer fires, then the virtual device driver releases the hardware. In this case, the virtual device driver does nothing when the virtual DMA device driver later calls with VDD_DMA_MASKOFF.

It is the virtual device driver's responsibility (in DMAHandlerFunc) to handle the unavailability of hardware ownership. This means the virtual device driver must put up the pop-up and suspend (or terminate) the DOS Session. The virtual device driver passes the interrupt notification to the virtual DMA device driver before simulating that interrupt to the DOS Session, if it owns the hardware.

In between VDD_DMA_MASKOFF and VDD_DMA_MASK events, the virtual DMA device driver always returns the physical DMA state, if the application polls its status or transfer count. This is appropriate because the virtual device driver owns the device during this period. VDD_DMA_MASK event notification takes place before actually masking the register in DMA. VDD_DMA_MASKOFF takes place after masking off the DMA.

DMA Channel 4 is not supported.

VDHRegisterProperty

This function registers a virtual device driver property with the DOS Session Manager.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHRegisterProperty:NEAR
PropertyName       DD     ?     ; Property name pointer
Reserved           DD     ?     ; Must be set to 0
Reserved           DD     ?     ; Must be set to 0
PropertyType       DW     ?     ; Property type
PropertyOrdinal    DW     ?     ; Property ordinal
PropertyFlag       DD     ?     ; Property flag
DefaultValue       DD     ?     ; Default value
ValidationData     DD     ?     ; Validation data structure
ValidationFunc     DD     ?     ; User-defined validation function

PUSH   PropertyName              ; Push the parameters
PUSH   Reserved                  ;
PUSH   Reserved                  ;
PUSH   PropertyType              ;
PUSH   PropertyOrdinal           ;
PUSH   PropertyFlag              ;
PUSH   DefaultValue              ;
PUSH   ValidationData            ;
PUSH   ValidationFunc            ;

CALL   VDHRegisterProperty       ; Call the function.
                                 ; Notice that it is not necessary to pop the variable pushed;
                                 ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
PropertyName	DD	Pointer to an ASCIIZ string containing the property name. Maximum length is 40 characters.
Reserved	DD	Must be set to *0*.
Reserved	DD	Must be set to *0*.
PropertyType	DW	Property type. See below.
PropertyOrdinal	DW	Property ordinal. See below.
PropertyFlag	DD	Property flag. Possible value: **VDMP_CREATE** Property can be specified only at DOS Session creation. Any change to the property after DOS Session creation is ignored.
DefaultValue	DD	Default value of the property. See below.
ValidationData	DD	Validation data. See below.
ValidationFunc	DD	Function which validates and accepts changes to this property for a running DOS Session. Ignored if VDMP_CREATE is specified in conjunction with VDMP_BOOL, VDMP_INT, or VDMP_ENUM.

PropertyType Property type. Values are:

VDMP_BOOL	0—Boolean
VDMP_INT	1—Integer. DD size, but only DW is valid
VDMP_ENUM	2—Enumeration
VDMP_STRING	3—ASCIIZ string
VDMP_MLSTR	4—Multi-line string, separated by linefeed (0AH).

PropertyOrdinal Property ordinal. Values are:

VDMP_ORD_OTHER	0—Custom virtual device driver property
VDMP_ORD_KERNEL	1—ASCIIZ path of DOS kernel
VDMP_ORD_SHELL	2—ASCIIZ path of DOS_SHELL
VDMP_ORD_RMSIZE	3—Integer size of DOS box (128KB—640KB)
VDMP_ORD_FCB	4—Integer total FCBs
VDMP_ORD_FCB2	5—Integer FCBs immune to close LRUing
VDMP_ORD_BREAK	6—Boolean Break flag
VDMP_ORD_DOSDD	7—Multi-line string DOS_DEVICE
VDMP_ORD_VMBOOT	8—ASCIIZ string virtual machine boot drives
VDMP_ORD_VERSION	10—Multi-line string fake version entries
VDMP_ORD_DOS_UMB	11—Boolean flag. DOS_UMB
VDMP_ORD_DOS_HIGH	12—Boolean flag. DOS_HIGH
VDMP_ORD_LASTDRIVE	13—ASCIIZ DOS_LASTDRIVE
VDMP_ORD_FILES	14—Integer total FILES.

DefaultValue Default value of the property. The format of this field depends on the value of PropertyType:

VDMP_BOOL	DefaultValue is interpreted as a BOOL (Boolean) value.
VDMP_INT	DefaultValue is interpreted as a DD value. Only the low half is used (the high half is ignored) so this is more similar to a DW than a DD.
VDMP_ENUM	DefaultValue is interpreted as a pointer to an ASCIIZ string.
VDMP_STRING	DefaultValue is interpreted as a pointer to an ASCIIZ string.
VDMP_MLSTR	DefaultValue is interpreted as a pointer to an ASCIIZ string.

ValidationData Validation data. The value of this field depends on the value of PropertyType:

VDMP_BOOL ValidationData is NULL. The user is expected to validate Booleans.

VDMP_INT ValidationData is a pointer to a VPBOUND structure:

VPBOUND Limits for VDMP_INT properties. Notice that max > min must hold, max—min must be a multiple of step, and step=*1* implies that all values between min and max are valid:

```
VPBOUND_s STRUC
    min    DW   ?   ; minimum allowed value
    max    DW   ?   ; maximum allowed value
    step   DW   ?   ; increment between values
VPBOUND_s ENDS
```

VDMP_ENUM ValidationData is a pointer to a set of ASCIIZ strings terminated by a zero byte, which is the allowed set of responses. The Shell uses this to construct a *list* or *combination* box for the user to pick from. Empty ("\0") strings are not allowed.

VDMP_STRING ValidationData is a DD that is the maximum allowed string length (including the NULL terminator).

VDMP_MLSTR ValidationData is a DD that is the maximum allowed string length (including the NULL terminator).

The interface for the ValidationFunc is defined as follows:

```
VDHPROP_VALIDATE  EQU  0x00000000L
VDHPROP_SET       EQU  0x00000001L

ValidationFunc  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 16] - op (DD)
;           [ESP + 12] - hvdm (DD)
;           [ESP +  8] - cb (DD)
;           [ESP +  4] - pch (DD)
```

PFNVDHRP is a virtual device driver property function that performs property setting and validation. Set operations can be requested any time after a DOS Session is created. The validation operation can be requested at any time, even before a DOS Session is created. Validation is requested only for VDMP_STRING and VDMP_MLSTR types because all other types can be validated using the information supplied by VDHRegisterProperty.

Parameter	Data Type	Description
op	DD	Operation to perform. See below.
hvdm	DD	Handle of DOS Session. Undefined, if op = VDHPROP_VALIDATE.
cb	DD	Count of bytes pointed to by pch. See below.
pch	DD	Pointer to the value to set or validate. See below.

op Operation to perform (enumeration):

VDHPROP_VALIDATE Validate property for any process. Only called for VDMP_STRING and VDMP_MLSTR properties.

VDHPROP_SET Set an already validated property for specified *hvdm*. The return code is ignored.

cb Count of bytes pointed to by pch. Value depends upon PropertyType:

VDMPROP_BOOL Undefined
VDMPROP_INT Undefined
VDMPROP_ENUM Length of ASCIIZ string including NULL terminator
VDMPROP_STRING Length of ASCIIZ string including NULL terminator
VDMPROP_MLSTR Length of ASCIIZ string including NULL terminator.

pch Value to set/validate. The format depends on the proptype:

VDMPROP_BOOL pch is interpreted as a BOOL (Boolean). Value *0* is FALSE, a non-zero value is TRUE.

VDMPROP_INT pch is interpreted as a DD, and is guaranteed to meet the registered bounds.

VDMPROP_ENUM pch points to an ASCIIZ string, and is guaranteed to be one of the registered enumeration strings.

VDMPROP_STRING pch points to an ASCIIZ string, and is guaranteed to be less than, or equal to, the registered maximum string length.

VDMPROP_MLSTR pch points to an ASCIIZ string. Multiple lines are separated by a line feed (0x0A). It is guaranteed to be less than, or equal to, the registered maximum string length.

If PFNVDHRP is successful, it returns *0*. If PFNVDHRP fails or if the value is invalid, this function returns the non-zero error code, ERROR_INVALID_DATA. PFNVDHRP can be called only in the task context.

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, it returns 0. VDHGetError should be called to determine the nature of the problem. *A system halt occurs if*:

- Any of the pointers (PropertyName, HelpFile, ValidationData, or ValidationFunc) are invalid
- PropertyFlag or PropertyType are invalid
- A VPBOUND structure has invalid contents
- The maximum string length for a VDMP_STRING is less than the length of the default string value.

Remarks:

Context Issues: This function can be called only in the initialization context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: VDHQueryProperty is used to obtain the value of a virtual device driver property for a particular DOS Session.

VDHRegisterVDD

This function registers virtual device driver entry points for use by other virtual device drivers (through VDHOpenVDD and VDHRequestVDD), and by OS/2 applications (through DosOpenVDD and DosRequestVDD).

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHRegisterVDD:NEAR
VDDName       DD     ?       ; Pointer to the name of the VDD
DosReqFcn     DD     ?       ; Function for use by DosRequestVDD
VDDReqFcn     DD     ?       ; Function for use by VDHRequestVDD

PUSH   VDDName               ; Push the parameters
PUSH   DosReqFcn             ;
PUSH   VDDReqFcn             ;

CALL   VDHRegisterVDD        ; Call the function
                             ; Notice that it is not necessary to pop the variable pushed;
                             ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
VDDName	DD	Pointer to the name of the virtual device driver. Maximum length is MAXLENXDD.
DosReqFcn	DD	Function used by DosRequestVDD. NULL, if none. See function structure below.
VDDReqFcn	DD	Function used by VDHRequestVDD. NULL, if none. See function structure below.

The interface for the DosReqFcn is:

```
DosReqFcn  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 24] - SGid    (DW) - Screen Group ID
;           [ESP + 20] - ulCmd   (DD) - Command
;           [ESP + 16] - cbIn    (DD) - Input buffer size
;           [ESP + 12] - pReqIn  (DD) - Input packet
;           [ESP +  8] - cbOut   (DD) - Output buffer size
;           [ESP +  4] - pReqOut (DD) - Output packet
;    EXIT   VDDREQ_PASS  - DOS Session manager calls the next routine
;                          registered under same name.
;           Non-zero     - Return failure (For APIs, 0 is success)
;           0            - Return the caller success
```

The interface for the VDDReqFcn is:

```
VDDReqFcn  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 16] - hvdm (DD)    - DOS Session handle
;           [ESP + 12] - ulCmd (DD)   - Command
;           [ESP +  8] - pReqIn (DD)  - Input packet
;           [ESP +  4] - pReqOut (DD) - Output packe
;    EXIT   VDDREQ_PASS  - DOS Session manager calls the next routine
;                          registered under same name.
;           Non-zero     - Return success (For Virtual DevHlp services, 1 is success)
;           0            - Return the caller failure
```

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, it returns 0. If this function is called with invalid parameters or in an incorrect context, *a system halt occurs.*

Remarks:

Context Issues: This function can be called only in the initialization context.

DOS Session Terminations: The virtual device driver must communicate with any virtual device driver or OS/2 clients, to ensure that the resources for the terminating DOS Session are freed.

Notes: If a virtual device driver fails in its INIT routine after registering (through VDHRegisterVDD), the de-registration is done automatically at the time the virtual device driver is unloaded. Because two or more virtual device drivers can register the same name (VDDName), the DOS Session Manager calls each virtual device driver's routine in turn, until one of them returns a non-zero value.

Notice that the order of the calling sequence in not consistent.

VDHReleaseCodePageFont

This function releases a code page font loaded with VDHGetCodePageFont. If system memory was allocated to hold the specified font, this call will free that memory.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHReleaseCodePageFont:NEAR
FontPtr     DD     ?              ; Pointer returned by VDHGetCodePageFont

PUSH   FontPtr                    ; Push the parameter

CALL   VDHReleaseCodePageFont     ; Call the function
                                  ; Notice that it is not necessary to pop the variable pushed;
                                  ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
FontPtr	DD	Pointer returned in FontAddress by VDHGetCodePageFont.

Results

Success If the function is successful, it returns nothing.

Failure If FontPtr is not valid, *a system halt occurs.*

Remarks:

Context Issues: This function can be called only in the DOS Session task context.

DOS Session Terminations: The virtual device driver must call this function for any code page fonts loaded for this DOS Session.

VDHReleaseMutexSem

This function releases the ownership of a mutex semaphore. If the request count becomes *0*, the highest priority semaphore is awakened.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHReleaseMutexSem:NEAR
MutexSemHandle    DD    ?    ; Handle to a mutex semaphore

PUSH   MutexSemHandle         ; Push the parameter

CALL   VDHReleaseMutexSem     ; Call the function
                              ; Notice that it is not necessary to pop the variable pushed;
                              ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
MutexSemHandle	DD	Handle of the semaphore to be released.

Results

Success — If the function is successful, it returns nothing and the semaphore in MutexSemHandle is released.

Failure — If the virtual device driver that called VDHReleaseMutexSem is not the owner of the semaphore to be released, or if MutexSemHandle is invalid, *a system halt occurs*.

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHRemoveFaultHook

This function removes the page fault handler hook for the specified DOS Session page range.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHRemoveFaultHook:NEAR
VDMHandle                 DD    ?      ; DOS Session handle.  0 => current DOS Session
StartingAddress           DD    ?      ; Starting linear address
Pages                     DD    ?      ; Number of pages
PageFaultHandlerFcnPtr    DD    ?      ; Function supplied to VDHInstallFaultHook

PUSH   VDMHandle                       ; Push the parameters
PUSH   StartingAddress                 ;
PUSH   Pages                           ;
PUSH   PageFaultHandlerFcnPtr          ;

CALL   VDHRemoveFaultHook              ; Call the function
                                       ; Notice that it is not necessary to pop the variable
                                       ; pushed; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
VDMHandle	DD	DOS Session handle. A value of *0* indicates the current DOS Session.
StartingAddress	DD	Starting linear address.
Pages	DD	Number of pages.
PageFaultHandlerFcnPtr	DD	Function supplied to VDHInstallFaultHook. This is used to verify that the calling virtual device driver is the one that installed the fault hook.

Results

Success If the function is successful, it returns nothing.

Failure If any of the input parameters are invalid, *a system halt occurs.*

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: Any fault hooks for the terminating DOS Session must be released.

VDHRemoveIOHook

This function must be called with identical StartingPort and NumPorts to remove the I/O hooks for the specified I/O ports. Port hooks cannot be removed for a subset of the range of ports hooked by VDHInstallIOHook.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHRemoveIOHook:NEAR
Reserved         DD     0     ; Reserved.  Must be set to 0.
StartingPort     DD     ?     ; Starting port number
NumPorts         DD     ?     ; Number of ports in the range
IOPortHook       DD     ?     ; Pointer to installed I/O hook entry

PUSH   Reserved               ; Push the parameters
PUSH   StartingPort           ;
PUSH   NumPorts               ;
PUSH   IOPortHook             ;

CALL   VDHRemoveIOHook        ; Call the function
                              ; Notice that it is not necessary to pop the variable pushed;
                              ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
Reserved	DD	Reserved. Must be set to 0.
StartingPort	DD	Starting port number.
NumPorts	DD	Number of ports in the range.
IOPortHook	DD	Pointer to installed I/O hook entry. This parameter is used to verify that the calling virtual device driver is the one that installed the I/O hook.

Results

Success If the function was successful, it returns nothing.

Failure If IOPortHookEntry is invalid or if StartingPort or NumPorts are out of range, *a system halt occurs.*

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: There are no DOS Session termination implications for this function. I/O hooks are automatically removed during DOS Session termination.

Notes: If the IOPortHook is in instance data, the address passed to VDHInstallIOHook must be the same address passed to VDHRemoveIOHook or VDHSetIOHookState.

VDHReportPeek

This function reports DOS Session polling activity. A counter of idle polling activity is incremented. If the count exceeds a threshold, the current DOS Session is put to sleep for a period.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHReportPeek:NEAR
PeekWeight     DD     ? ; Value to add to the idle counter

PUSH   PeekWeight       ; Push the parameter

CALL   VDHReportPeek    ; Call the function
                        ; Notice that it is not necessary to pop the parameter pushed;
                        ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
PeekWeight	DD	Value to add to the idle counter.

Results

Success If the function is successful, it returns nothing.

Failure If the current process is not a DOS Session, *a system halt occurs.*

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: Any virtual device driver that can detect idle polling activity can call this service to report it. If the sum of peek weights exceeds 64KB in a single VDHGSV_MSECSBOOT clock tick (see "VDHQuerySysValue" on page 5-113), the DOS Session is considered idle. It is the responsibility of the virtual device drivers to calibrate its peek weight. This depends on machine speed and the Ring 0 trap overhead time, and therefore cannot be calibrated from measuring peek rate under DOS.

Notice that testing the time before the count ensures that new tests begin at the first peek in a time slice. This allows use of a very short test period.

VDHRequestMutexSem

This function requests the ownership of a mutex semaphore. If the semaphore is owned and the caller is not the owner, the thread will block. If the caller is the owner, a request count is incremented. Request calls can be nested. A maximum of 65,535 (64KB) requests are allowed for each semaphore at any one time. Exceeding this limit *results in a system halt*.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHRequestMutexSem:NEAR
MutexSemHandle      DD     ?       ; Mutex semaphore handle
Timeout             DD     ?       ; Timeout (in milliseconds)

PUSH   MutexSemHandle              ; Push the parameters
PUSH   Timeout                     ;

CALL   VDHRequestMutexSem          ; Call the function
                                   ; Notice that it is not necessary to pop the variable pushed;
                                   ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
MutexSemHandle	DD	Mutex semaphore handle
Timeout	DD	Number of milliseconds to wait before timing out the semaphore request.

Results

Success If the function is successful, it returns *1*.

Failure If the function fails, it returns *0*. VDHGetError should be called to determine the nature of the problem. An invalid semaphore *will result in a system halt*. A system halt also occurs if the request count crosses the 65,535 (64KB) limit.

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHRequestVDD

This function requests an operation of a virtual device driver. This service is also used for communication between two or more virtual device drivers where dynamic linkage between them is not appropriate.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHRequestVDD:NEAR
VDDHandle             DD   ?    ; Virtual device driver handle
VDMHandle             DD   ?    ; DOS Session handle
Command               DD   ?    ; Command requested
InputRequestPacket    DD   ?    ; Pointer to the input packet
OutputRequestPacket   DD   ?    ; Pointer to the output packet

PUSH   VDDHandle                ; Push the parameters
PUSH   VDMHandle                ;
PUSH   Command                  ;
PUSH   InputRequestPacket       ;
PUSH   OutputRequestPacket.     ;

CALL   VDHRequestVDD            ; Call the function
                                ; Notice that it is not necessary to pop the variable pushed;
                                ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
VDDHandle	DD	Handle to a virtual device driver
VDMHandle	DD	Handle to a DOS Session
Command	DD	Command requested
InputRequestPacket	DD	Pointer to the input packet
OutputRequestPacket	DD	Pointer to the output packet.

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, it returns 0. VDHGetError should be called to determine the nature of the problem. If VDDHandle or VDMHandle is invalid, *a system halt occurs*.

Remarks:

Context Issues: This function can be called in the task or the interrupt context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: Every VDHRequestVDD procedure registered under the virtual device driver name associated with the given handle is called until one returns a non-zero value. No assumption should be made about the order to the calling sequence.

The virtual device driver worker routine sets the error with VDHSetError when it returns 0. If the calling virtual device driver is registered under the same name as the virtual device driver handle, then its entry point is also called. Furthermore, the DOS Session Manager does not prevent any of the called virtual device drivers from issuing another VDHRequestVDD.

VDHReservePages

This function reserves a range of linear addresses for later use with VDHMapPages or VDHAllocPages. A reserved area cannot contain a mixture of pages set by VDHMapPages and VDHAllocPages, but it can be used successively with either one.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHReservePages:NEAR
StartingAddress     DD   ?    ; Starting address of the linear memory to reserve
NumPages            DD   ?    ; Number of pages to reserve

PUSH   StartingAddress        ; Push the parameters
PUSH   NumPages               ;

CALL   VDHReservePages        ; Call the function
                              ; Notice that it is not necessary to pop the variable pushed;
                              ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
StartingAddress	DD	Starting address of the linear memory to reserve. Must be page aligned and must be less than 110000H (1MB + 64KB).
NumPages	DD	Number of pages to reserve.

Results

Success — If the function is successful, it returns a non-zero value and the pages are reserved for later access.

Failure — If the function fails, it returns 0. VDHGetError should be called to determine the nature of the problem. If StartingAddress or NumPages is invalid, *a system halt occurs*.

Remarks:

Context Issues: This function can be called in the initialization and DOS Session-task contexts. In the initialization context, the reservation affects only the global linear memory map. In the DOS Session-task context, reservations affect only the local linear memory map.

DOS Session Terminations: There are no DOS Session termination implications for this function. The DOS Session Manager will clean up these reservations when the DOS Session terminates.

VDHResetEventSem

This function resets an event semaphore.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHResetEventSem:NEAR
EventSemHandle     DD     ?       ; Handle to an event semaphore

PUSH   EventSemHandle             ; Push the parameter

CALL   VDHResetEventSem           ; Call the function
                                  ; Notice that it is not necessary to pop the variable pushed;
                                  ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
EventSemHandle	DD	Handle to the event semaphore to reset.

Results

Success If the function is successful, it returns nothing and the semaphore is reset.

Failure Resetting a semaphore that is invalid or is already reset *causes a system halt to occur*.

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHSeek

This function seeks to a specified position within a file previously opened by VDHOpen.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHSeek:NEAR
FileHandle      DW    ?     ; Handle to the file to seek in
NewOffset       DD    ?     ; Offset of the new file pointer position
MoveTypeFlag    DD    ?     ; Type of move (absolute, relative)

PUSH   FileHandle           ; Push the parameters
PUSH   NewOffset            ;
PUSH   MoveTypeFlag         ;

CALL   VDHSeek              ; Call the function.
                            ; Notice that it is not necessary to pop the variable pushed;
                            ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
FileHandle	DW	Handle to the file
NewOffset	DD	Offset to the new file-pointer position. Depending on the value of MoveTypeFlag, this can be the offset from the beginning of the file, the ending of the file, or the current position of the file-pointer.
MoveTypeFlag	DD	Indicates what type of move is being done. See below.

MoveTypeFlag Indicates what type of move is being done. Possible values are:

VDHSK_ABSOLUTE	Move file-pointer to a location relative to the beginning of the file
VDHSK_CURRENT_POSITION	Move file-pointer relative to its current position
VDHSK_END_OF_FILE	Move file-pointer to a location relative to the end of the file.

Results

Success If the function is successful, it returns the new current absolute position of the file-pointer within the file.

Failure If the function fails, it returns 0FFFFFFFFH. VDHGetError should be called to determine the nature of the problem. If FileHandle is invalid, or if this function is not called in the DOS Session-task context, *a system halt occurs.*

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHSendVEOI

This function sends a virtual End-Of-Interrupt (EOI) signal to the Virtual Programmable Interrupt Controller (VPIC), and clears the in-service state of the IRQ for the calling device. The EOI handler for the IRQ is called, if one exists.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHSendVEOI:NEAR
IRQHandle     DD     ?        ; IRQ handle from VDHOpenVIRQ

PUSH   IRQHandle              ; Push the parameter

CALL   VDHSendVEOI            ; Call the function
                              ; Notice that it is not necessary to pop the variable pushed;
                              ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
IRQHandle	DD	IRQ handle.

Results

Success If the function was successful, it returns nothing.

Failure If IRQHandle is invalid, *a system halt occurs.*

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHSetA20

This function enables or disables the A20 state for the current DOS Session.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHSetA20:NEAR
StateFlag     DD     ?        ; Indicates whether A20 state is enabled or disabled

PUSH   StateFlag              ; Push the parameter

CALL   VDHSetA20              ; Call the function
                              ; Notice that it is not necessary to pop the variable pushed;
                              ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
StateFlag	DD	Indicates whether the A20 line is enabled or disabled: **Non-zero** Enable A20 line, wrapping is *off*. **0** Disable A20 line, wrapping is *on*.

Results: None.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHSetDosDevice

This function links a DOS device driver into the chain of DOS device drivers for a DOS Session.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHSetDosDevice:NEAR
vpDosDD     DD     ?          ; V86 FAR address of the DOS device driver header

PUSH   vpDosDD                ; Push the parameter

CALL   VDHSetDosDevice        ; Call the function
                              ; Notice that it is not necessary to pop the variable pushed;
                              ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
vpDosDD	DD	V86 FAR address of the DOS device driver header. This header (and any headers chained to it) are entered into the DOS device driver list for this DOS Session.

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, it returns 0.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context (DOS Session creation only).

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHSetError

This function sets the error code for return by VDHGetError.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHSetError:NEAR
ErrorCode    DD    ?          ; Error code to set

PUSH   ErrorCode              ; Push the parameter

CALL   VDHSetError            ; Call the function
                              ; Notice that it is not necessary to pop the variable pushed;
                              ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
ErrorCode	DD	Error code to set for VDHGetError.

Results: None.

Remarks:

Context Issues: This function can be called in any context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: This function is intended for use by any virtual device driver that offers dynamic link (dynalink) services to other virtual device drivers. Most of the base errors are defined in BSEERR.INC. Where possible, these definitions are used when returning errors.

VDHSetFlags

This function sets the DOS Session's Flags Register to the specified values.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHSetFlags:NEAR
FlagValue     DD     ?        ; DOS Session flag value

PUSH   FlagValue              ; Push the parameter

CALL   VDHSetFlags            ; Call the function
                              ; Notice that it is not necessary to pop the variable pushed;
                              ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
FlagValue	DD	DOS Session flag value that sets the DOS Session's Flags Register.

Results: None.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: Virtual device drivers must use this interface instead of the client register frame pointer to change the DOS Session's flags. Changes to the interrupt flag and the I/O Privilege Level (IOPL) field must be under the control of 8086 emulation so that VDHArmSTIHook works correctly.

Nested Task (NT) and Resume Flag (RF) flags are cleared, the Virtual 8086 Mode (VM) flag is set, and the IOPL field is left unchanged. This function does not take effect until the operating system returns to V86 mode.

VDHSetIOHookState

This function is used to enable and disable I/O port trapping for a range of ports.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHSetIOHookState:NEAR
Reserved         DD     0        ; Reserved.  Must be set to 0.
StartingPort     DD     ?        ; Starting port number
NumPorts         DD     ?        ; Number of ports in the range
IOPortHook       DD     ?        ; Pointer used to install I/O hooks
EnableFlag       DD     ?        ; If !0, enable I/O hooks; if 0, disable I/O hooks

PUSH   Reserved                  ; Push the parameters
PUSH   StartingPort              ;
PUSH   NumPorts                  ;
PUSH   IOPortHook                ;
PUSH   EnableFlag                ;

CALL   VDHSetIOHookState         ; Call the function
                                 ; Notice that it is not necessary to pop the variable pushed;
                                 ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
Reserved	DD	Reserved. Must be set to *0*.
StartingPort	DD	Starting port number.
NumPorts	DD	Number of ports in the range.
IOPortHook	DD	Pointer used to install I/O hooks. This is used to verify that the calling virtual device driver is the one that installed the I/O hooks.
EnableFlag	DD	Enable/Disable Flag. Possible values are: **TRUE (non-zero)** Enable I/O hooks **FALSE (0)** Disable I/O hooks.

Results

Success If the function was successful, it returns nothing.

Failure If IOPortHook is invalid, or if StartingPort or NumPorts are out of range, or if StartingPort and NumPorts overlap a range of ports previously marked as VDHIIH_ALWAYS_TRAP by VDHInstallIOHook, *a system halt occurs.*

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: If the IOPortHook is in instance data, the address passed to VDHInstallIOHook must be the same address passed to VDHRemoveIOHook or VDHSetIOHookState. When trapping is enabled, the I/O port hooks for the specified range get control when a DOS Session does I/O to that range. When trapping is disabled, DOS Session I/O goes directly to the physical hardware ports. None of the ports in the range can be marked as VDHIIH_ALWAYS_TRAP by VDHInstallIOHook.

VDHSetPriority

This function adjusts a DOS Session's scheduler priority class and level. Priority levels within each priority class range from *0–31*.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHSetPriority:NEAR
VDMHandle           DD    ?     ; Handle to the DOS Session with the priority to change
ActionClassFlag     DD    ?     ; Dual purpose flag, Action and Class
NewPriority         DD    ?     ; The change from the current priority

PUSH   VDMHandle                ; Push the parameters
PUSH   ActionClassFlag          ;
PUSH   NewPriority              ;

CALL   VDHSetPriority           ; Call the function
                                ; Notice that it is not necessary to pop the variable pushed;
                                ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
VDMHandle	DD	Handle to the DOS Session with the priority to change
ActionClassFlag	DD	A dual-purpose flag. See below.
NewPriority	DD	Change from the DOS Session's previous priority level.

ActionClassFlag A dual-purpose flag. The Class component of ActionClassFlag indicates which class to change the DOS Session to. The Action component of the ActionClassFlag allows the virtual device driver to manage when the priority is changed, and when to start, end, or continue use of the class. See "Notes" below.

The values of the Class component can be one of the following:

VDHSP_TIME_CRITICAL	Highest priority
VDHSP_SIMULATED_INTERRUPT	•
VDHSP_SERVER	•
VDHSP_REGULAR	•
VDHSP_IDLE	Lowest priority
VDHSP_NO_CHANGE	Do not change the DOS Session's class.

The values of the Action component can be one of the following:

VDHSP_START_USE	Start use of a class
VDHSP_CONTINUE_USE	Continue use of a class
VDHSP_END_USE	End use of a class
VDHSP_DEFAULT_ACTION	If no action is specified, the default class is changed.

Results

Success If the function is successful, it returns nothing.

Failure If VDMHandle or ActionClassFlag are invalid, *a system halt occurs.*

Remarks:

Context Issues: This function can be called in the task or interrupt contexts.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: A count of the current users is maintained at each class. The actual priority class in effect at any time is always the highest priority class with current declared users. When there are no declared users, the default class is used. Changes to a class other than the effective class are made to saved values. These changes take effect when the class becomes the effective class.

The caller must signal when it begins use of the class, when it makes changes within the class, and when it finishes with the class. The typical user would make one or more changes to priority at a particular class, and finish the series of changes at the class. The virtual device driver uses the VDHSP_START_USE and VDHSP_END_USE flags at the first and last calls within the class, and the VDHSP_CONTINUE_USE flag for changes in between.

The default class is normally VDHSP_REGULAR unless the class or NewPriority is changed without declaring a user (that is, without specifying a start, continue, or end action flag). This allows a user interface to adjust the default priority of a DOS Session.

VDHSetVIRR

This function sets the virtual Interrupt Request Register (IRR) in the Virtual Programmable Interrupt Controller (VPIC) of the specified DOS Session for the IRQ specified. This causes an interrupt to be simulated to the DOS Session.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHSetVIRR:NEAR
VDMHandle     DD     ?        ; DOS Session handle
IRQHandle     DD     ?        ; Handle from VDHOpenVIRQ

PUSH   VDMHandle              ; Push the parameters
PUSH   IRQHandle              ;

CALL   VDHSetVIRR             ; Call the function
                              ; Notice that it is not necessary to pop the variable pushed;
                              ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
VDMHandle	DD	DOS Session handle. A value of *0* indicates the current DOS Session.
IRQHandle	DD	IRQ handle from VDHOpenVIRQ.

Results

Success If the function was successful, it returns nothing.

Failure If either of the parameters is invalid, *a system halt occurs.*

Remarks:

Context Issues: This function can be called in the task or interrupt context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: If the interrupt request has not been cleared when the DOS Session issues an End Of Interrupt (EOI), another interrupt will be simulated to the DOS Session.

VDHSetVPMExcept

This function sets the current value in the protect-mode exception table.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHSetVPMExcept:NEAR
Vector            DD    ?   ; Interrupt vector number
HandlerAddress    DQ    ?   ; Far32 handler address
Flag              DB    ?   ;

PUSH   Vector               ; Push the parameters
PUSH   HandlerAddress       ;
PUSH   Flag                 ;

CALL   VDHSetVPMExcept      ; Call the function
                            ; Notice that it is not necessary to pop the variable pushed;
                            ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
Vector	DD	Interrupt vector number
HandlerAddress	DQ	Far32 handler address
Flag	DB	Flag indicating what kind of exception handler is being registered. Possible value: **VPMXCPT32** A 32-bit handler is being registered.

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, it returns 0.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHSetVPMIntVector

This function sets the application Ring 3 handler in the protect-mode interrupt chain. This is used only for DOS Protect Mode Interface (DPMI) support.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHSetVPMIntVector:NEAR
Vector             DD    ?    ; Interrupt vector number
HandlerAddress     DQ    ?    ; Far32 handler address

PUSH   Vector                 ; Push the parameters
PUSH   HandlerAddress         ;

CALL   VDHSetVPMIntVector     ; Call the function
                              ; Notice that it is not necessary to pop the variable pushed;
                              ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
Vector	DD	Interrupt vector number
HandlerAddress	DQ	Far32 handler address.

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, it returns 0.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHSwitchToVPM

This function switches the current DOS Session into protect mode. This assumes that the appropriate initializations have already been performed.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHSwitchToVPM:NEAR

CALL   VDHSwitchToVPM     ; Call the function
```

Parameters: None.

Results: None.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHSwitchToV86

This function switches the current DOS Session to V86 mode.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHSwitchToV86:NEAR

CALL   VDHSwitchToV86     ; Call the function
```

Parameters: None.

Results: None.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHThawVDM

This function reverses the effect of VDHFreezeVDM. The specified DOS Session is allowed to run when the *freeze count* becomes *0*.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHThawVDM:NEAR
VDMHandle     DD     ?        ; Handle to the DOS Session to thaw

PUSH   VDMHandle              ; Push the parameter

CALL   VDHThawVDM             ; Call the function
                              ; Notice that it is not necessary to pop the variable pushed;
                              ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
VDMHandle	DD	Handle to the DOS Session to allow to execute (thaw).

Results

Success If the function is successful, it returns nothing. The freeze count for the DOS Session is decremented.

Failure If VDMHandle is an invalid handle, *a system halt occurs*.

Remarks:

Context Issues: This function can be called in the task or interrupt context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: This function does nothing if called on behalf of a DOS Session that is not frozen. See "VDHFreezeVDM" on page 5-52 for a full discussion of freeze counting.

VDHUnlockMem

This function reverses the effect of VDHLockMem, unlocking a previously locked area of memory.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHUnlockMem:NEAR
LockHandle     DD     ?      ; Lock handle from VDHLockMem

PUSH   LockHandle            ; Push the parameter

CALL   VDHUnlockMem          ; Call the function
                             ; Notice that it is not necessary to pop the variable pushed;
                             ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
LockHandle	DD	Lock handle of a locked memory area. Originally obtained from VDHLockMem.

Results

Success If the function is successful, it returns nothing.

Failure An invalid lock handle *causes a system halt*.

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: Because lock handles are global, this function can be made in any task context.

VDHUnreservePages

This function unreserves pages that were previously reserved with VDHReservePages. The starting address and size must be identical to the previous corresponding call to VDHReservePages. Notice that any mapping made earlier on this region is unmapped before calling this function.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHUnreservePages:NEAR
StartingAddress     DD    ?     ; Starting address of the region to unreserve
NumPages            DD    ?     ; Size of the region to unreserve, in pages

PUSH   StartingAddress          ; Push the parameters
PUSH   NumPages                 ;

CALL   VDHUnreservePages        ; Call the function
                                ; Notice that it is not necessary to pop the variable pushed;
                                ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
StartingAddress	DD	Starting address of the region to unreserve. Must be less than 110000H (1MB+64KB) and must be page aligned. In addition, StartingAddress must match a previous VDHReservePages call.
NumPages	DD	Size, in pages, of the region to unreserve. Must match the corresponding parameter to a previous call to VDHReservePages.

Results

Success If the function is successful, it returns nothing.

Failure If StartingAddress or NumPages is invalid, or if they were not used together in an earlier call to VDHReservePages, *a system halt occurs.*

Remarks:

Context Issues: This function can be called in the initialization or DOS Session-task context (DOS Session creation). In the initialization context, memory can be unreserved only from the global linear memory map. In the DOS Session-task context, memory can be unreserved only from the local linear memory map.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: StartingAddress must be less than 110000H(1MB+64KB) and must be page aligned. Additionally, StartingAddress must match a previous call to VDHReservePages. NumPages must match the corresponding parameter to a previous call to VDHReservePages.

VDHWaitEventSem

This function is used to wait on an event semaphore. If the semaphore is posted, it will return immediately.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHWaitEventSem:NEAR
EventSemHandle    DD   ?     ; Event semaphore handle
Timeout           DD   ?     ; Timeout (in milliseconds)

PUSH   EventSemHandle        ; Push the parameters
PUSH   Timeout               ;

CALL   VDHWaitEventSem       ; Call the function
                             ; Notice that it is not necessary to pop the variable pushed;
                             ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
EventSemHandle	DD	Event semaphore handle
Timeout	DD	Number of milliseconds to wait before timing out.

Results

Success If the function is successful, it returns *1*.

Failure If the function fails, it returns *0*. VDHGetError should be called to determine the nature of the problem. If EventSemHandle is invalid, *a system halt occurs*.

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: There are no DOS Session termination implications for this function. If Timeout is *0*, the caller will not block, however, ERROR_TIMEOUT is returned. If Timeout is *−1*, the caller is blocked until the semaphore is posted. Otherwise, the caller blocks for the time specified (in milliseconds) in Timeout.

VDHWaitVIRRs

This function waits until any virtual interrupt is simulated to the current DOS Session.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHWaitVIRRs:NEAR
PostIntFcnHook    DD   ?     ; Hook handle

PUSH   PostIntFcnHook        ; Push the parameter

CALL   VDHWaitVIRRs          ; Call the function
                             ; Notice that it is not necessary to pop the variable pushed;
                             ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
PostIntFcnHook	DD	Hook handle of a routine that is to be called after interrupts are simulated.

The interface for the routine is:

```
HookRoutine  PROC  NEAR
; PARAMETERS
;    ENTRY  [ESP + 8] - pRefData
;           [ESP + 4] - pcrf - client register frame pointer
;    EXIT   None
;
; USES      EAX, ECX, EDX, FS, GS, Flags
; CONTEXT   DOS Session task
```

Results

Success The function returns a non-zero value, if it woke up because of a simulated interrupt. If it woke up because of a VDHWakeVIRRs call, the function returns 0.

Failure If the function fails, it returns nothing.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: The hook handle must be allocated as a VDH_WAITVIRRS_HOOK type. This function is intended for virtual device drivers that need to simulate a spin loop without stopping virtual interrupt simulation. This allows the virtual device driver to be compatible with DOS but avoid wasting CPU time in a spin loop.

Generally, the virtual device driver will be in a loop when calling this function. When it returns a non-zero value, the virtual device driver returns to V86 mode so that the interrupt can be simulated. The hook function passed to this service is executed after all interrupts are simulated. When this function returns 0, it is because some other portion of the virtual device driver (likely in response to a physical hardware interrupt) has determined that the DOS Session no longer needs to spin. The virtual device driver exits the loop at this point.

The behavior described above is what the virtual keyboard device driver does in order to emulate the ROM BIOS Read Keyboard function (INT 16H, AH=00) without using excessive CPU time by allowing the ROM BIOS to spin out in V86 mode, or stopping simulated interrupts by performing a VDHWaitEventSem.

VDHWakeIdle

This function notes that the DOS Session is busy (doing useful work). If the DOS Session is currently sleeping or running at a lower priority because of the polling activities, the DOS Session is awakened, its priority is restored, and it is no longer considered idle.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHWakeIdle:NEAR
VDMHandle     DD     ?      ; Handle to the DOS Session to wake up

PUSH   VDMHandle            ; Push the parameter

CALL   VDHWakeIdle          ; Call the function
                            ; Notice that it is not necessary to pop the variable pushed;
                            ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
VDMHandle	DD	Handle to the DOS Session to wake up.

Results: This function returns nothing.

Remarks:

Context Issues: This function can be called in the task or interrupt context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHWakeVIRRs

This function wakes up a DOS Session that is waiting with the VDHWaitVIRRs service. See "VDHWaitVIRRs" on page 5-157 for a full description of the use of this function.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHWakeVIRRs:NEAR
VDMHandle     DD     ?        ; Handle of the DOS Session to wake up

PUSH   VDMHandle              ; Push the parameter

CALL   VDHWakeVIRRs           ; Call the function
                              ; Notice that it is not necessary to pop the variable pushed;
                              ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
VDMHandle	DD	Handle of the DOS Session to wake up.

Results

Success If the function was successful, it returns nothing.

Failure If VDMHandle is invalid, *a system halt occurs.*

Remarks:

Context Issues: This function can be called in the task or interrupt context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHWrite

This function writes bytes to a file or device previously opened by VDHOpen.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHWrite:NEAR
FileHandle          DW     ?     ; Handle to the file or device to write to
WriteBufferPtr      DD     ?     ; Pointer to the buffer to write
NumBytes            DD     ?     ; Number of bytes to write

PUSH   FileHandle                ; Push the parameters
PUSH   WriteBufferPtr            ;
PUSH   NumBytes                  ;

CALL   VDHWrite                  ; Call the function.
                                 ; Notice that it is not necessary to pop the variable pushed;
                                 ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
FileHandle	DW	Handle to the file or device to write to
WriteBufferPtr	DD	Pointer to the buffer to write
NumBytes	DD	Number of bytes to write.

Results

Success If the function is successful, it returns a count of the bytes written. This can be equal to *0*.

Failure If the function fails, it returns 0FFFFFFFFH. VDHGetError should be called to determine the nature of the problem. If FileHandle is invalid, or this function is called in any context except DOS Session-task, *a system halt occurs*.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

VDHWriteUBuf

This function writes to protect-mode address space as if access were done at Ring 3. This means checking for supervisor and read-only faults, and trapping any other faults. All faults are passed on to the DOS Session application handler through VDHRaiseException.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHWriteUBuf:NEAR
SourceBuffer      DD    ?     ; Source buffer to copy from
ByteCount         DD    ?     ; Count of bytes
Selector          DD    ?     ; Application selector
OffsetPointer     DD    ?     ; Address of the variable containing the offset
Flag              DD    ?     ; Flag for checking controls

PUSH   SourceBuffer           ; Push the parameters
PUSH   ByteCount              ;
PUSH   Selector               ;
PUSH   OffsetPointer          ;
PUSH   Flags                  ;

CALL   VDHWriteUBuf           ; Call the function
                              ; Notice that it is not necessary to pop the variable pushed;
                              ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
SourceBuffer	DD	Source buffer to copy from
ByteCount	DD	Count of bytes
Selector	DD	Application selector
OffsetPointer	DD	Address of the variable containing the offset for the start of the write
Flag	DD	Checking control. See below.

Flag Checking control. Possible values are:

VPM_PROT_READ	Check for read
VPM_PROT_WRITE	Check for write
VPM_FAULT_IF_SU_SET	Fault, if supervisor pages
VPM_FAULT_IF_RO	Fault, if the descriptor is read only
VPM_SEL_PRESENT	Caller knows descriptor is present
VPM_SEL_WRITEABLE	Caller knows descriptor is writable
VPM_SEL_IS_SS	Selector is client's stack
VPM_XCPTRET_ALT	After exception, return to alternate mode. For example, if the client was in protect mode when the service was called, return in V86 mode after the exception is handled.

Results

Success If the function is successful, it returns a non-zero value.

Failure If the function fails, it returns 0 if there is a bad address reference, which causes a fault. In such a case, OffsetPointer is updated with the address of the fault. For Selector Faults, OffsetPointer remains unchanged.

Remarks:

Context Issues: This function can be called only in the DOS Session-task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: When the routine fails, the caller must clean up and exit so that the exception can be simulated to the user.

VDHYield

This function yields the processor to any other thread of equal or higher priority.

Assembly Language Syntax

```
include mvdm.inc

EXTRN  VDHYield:NEAR
OptionFlag     DD     ?       ; Yield options

PUSH   OptionFlag             ; Push the parameter

CALL   VDHYield               ; Call the function
                              ; Notice that it is not necessary to pop the variable pushed;
                              ; the Virtual DevHlp function does this.
```

Parameters

Parameter	Data Type	Description
OptionFlag	DD	Yield options. Possible value: **VDH_YIELD_TIME_CRITICAL** Yield only to time critical threads.

Results: None.

Remarks:

Context Issues: This function can be called only in the task context.

DOS Session Terminations: There are no DOS Session termination implications for this function.

Notes: If OptionFlag is VDH_YIELD_TIME_CRITICAL and no time critical thread is able to run, the caller keeps the CPU.

Glossary

This glossary defines the terms used in this book. It includes terms and definitions from the *IBM Dictionary of Computing,* SC20-1699 as well as terms specific to the Presentation Manager but it is not a complete glossary for OS/2 Version 2.0.

A

ABIOS. Advanced BIOS. See *BIOS.*

action. One of a set of defined tasks a computer performs. Users request the application to perform an action in several ways: typing a command, pressing a function key or selecting the action name from an action bar or menu.

action point. The current position on the screen at which the pointer is pointing. (Contrast with *hotspot* and *input focus.*)

active program. A program currently running on the computer. See also *interactive program, noninteractive program* and *foreground program.*

active window. The window with which the user is currently interacting.

alphanumeric video output. Output to the logical video buffer when the video adapter is in text mode and the logical video buffer is addressed by an application as a rectangular array of character cells.

ANSI. American National Standards Institute.

APA. All points addressable.

API. Application programming interface. The formally defined programming language that is between an IBM application program and the user of the program. See also GPI.

ASCII. American National Standard Code for Information Interchange.

ASCIIZ. A string of ASCII characters that is terminated with a byte containing the value 0 (null).

aspect ratio. In computer graphics, the width-to-height ratio of an area, symbol or shape.

asynchronous. (1) Without regular time relationship. (2) Unexpected or unpredictable with respect to the execution of a program's instructions.

attributes. Characteristics or properties that can be controlled, usually to obtain a required appearance; for example, the color of a line. See also *segment attributes.*

AVIO. Advanced Video Input/Output.

B

background color. The color in which the background of a graphic primitive is drawn.

Basic Input/Output System. See *BIOS.*

Bezier curves. A mathematical technique of specifying smooth continuous lines and surfaces which require a starting point and a finishing point with several intermediate points that influence or control the path of the linking curve. Named after Dr. P. Bezier.

BIOS. Microcode that controls basic hardware operations (for example, interactions with hard disk drives, diskette drives, and the keyboard).

bitmap. A representation in memory of the data displayed on an APA device, usually the screen.

border. A visual indication (for example, a separator line or a background color) of the boundaries of a window.

button. A mechanism on a *pointing device,* such as a mouse, used to request or initiate an action. Contrast with *pushbutton* and *radio button.*

C

CASE statement. Provides, in C/2, the body of a window procedure. There is one CASE statement for each message type written to take specific actions.

CGA. Color graphics adapter.

chained list. Synonym for *linked list.*

character. A letter, digit or other symbol.

character code. The means of addressing a character in a character set, sometimes called *code point.*

check box. A control window shaped like a square button on the screen, that can be in a checked or unchecked state. It is used to select one or more items from a list. Contrast with *radio button.*

check mark. The symbol (√) that is used to indicate a selected item on a pull-down.

child window. A window that is positioned relative to another window (either a main window or another child window). Contrast with *parent window.*

choice. An option that can be selected. The choice can be presented as text, as a symbol (number or letter) or as an icon (a pictorial symbol).

class. See *window class*.

class style. The set of properties that apply to every window in a window class.

client area. The area in the center of a window that contains the main information of the window.

clipboard. An area of main storage that can hold data being passed from one Presentation Manager application to another. Various data formats can be stored.

clipping. In computer graphics, removing those parts of a display image that lie outside a given boundary.

clip limits. The area of the paper that can be reached by a printer or plotter.

code page. An assignment of graphic characters and control function meanings to all code points.

code point. Synonym for *character code*.

command. (1) The name and parameters associated with an action that can be performed by a program. A command is one form of action request. Users type in the command and enter it. (2) An action users request to interact with the command area.

command area. An area composed of two elements: a command field prompt and a command entry field.

command entry field. An entry field in which users type commands. The entry field is preceded by a command field prompt. These two elements make up the command area.

command line. On a display screen, a display line (usually at the bottom of the screen) in which only commands can be entered.

command prompt. A field prompt showing the location of the command entry field in a panel.

Common Programming Interface. A consistent set of specifications for languages, commands and calls to enable applications to be developed across all SAA environments. See also *Systems Application Architecture*.

Common User Access. A set of rules that define the way information is presented on the screen and the techniques for the user to interact with the information.

Compatibility Kernel. Portion of IBM Operating System/2 kernel that exists to support DOS INT 20, 21, 25, 26 ,27 functionality; acts as interface to common kernel functionality, for example, file system.

Contention. State where a DOS Session attempts to access a serially reusable resource (for example, a COM device) when another DOS Session or an IBM Operating System/2 application is using the resource.

Context Hook. Similar to force flags in IBM Operating System/2, these are events, signaled by a virtual device driver, that are processed at task time. Forcing an IRET and simulating an NMI can fall into this category.

control. The means by which an operator gives input to an application. A *choice* corresponds to a control.

Control Panel. In the Presentation Manager, a program used to set up user preferences that act globally across the system.

Control Program. The basic function of OS/2 including DOS emulation and the support for keyboard, mouse and VIO.

control window. A class of window used to handle a specific kind of user interaction. Radio buttons and check boxes are examples.

CPI. See *Common Programming Interface*.

cursor. A symbol displayed on the screen and associated with an input device. The cursor indicates where input from the device will be placed. Types of cursors include text cursors, graphics cursors and selection cursors. Contrast with *pointer* and *input focus*.

D

data structure. (ISO) The syntactic structure of symbolic expressions and their storage allocation characteristics.

DBCS. See *double-byte character set*.

default value. A value used when no value is explicitly specified by the user. For example, in the graphics programming interface, the default line type is 'solid'.

Desktop Manager. In the Presentation Manager, a window from which users can start one or more listed programs.

desktop window. The window corresponding to the physical device against which all other types of windows are established.

device context. A logical description of a data destination such as memory, metafile, display, printer or plotter. See also *direct device context*, *information device context*, *memory device context*, *metafile device context*, *queued device context* and *screen device context*.

device driver. A file that contains the code needed to use a device such as a display, printer or plotter.

Device Helpers (DevHlp). Kernel services (memory, hardware interrupt, software interrupt, queuing, semaphore,...) provided to physical device drivers.

dialog. The interchange of information between a computer and its user through a sequence of requests by the user and the presentation of responses by the computer.

dialog box. A type of window that contains one or more controls for the formatted display and entry of data. Also known as a *pop-up window*. A modal dialog box is used to implement a pop-up window.

Dialog Box Editor. A what-you-see-is-what-you-get (WYSIWYG) editor that creates dialog boxes for communicating with the application user.

dialog item. A component (for example, a menu or a button) of a dialog box. Dialog items are also used when creating dialog templates.

dialog template. The definition of a dialog box which contains details of its position, appearance and window ID; and the window ID of each of its child windows.

direct manipulation. The action of using the mouse to move objects around the screen. For example, moving files and directories about in the File Manager.

directory. A type of file containing the names and controlling information for other files or other directories.

DOS Session. A session created by the OS/2 Operating System that supports the independent execution of a DOS program. The DOS program appears to run independent of any other program in the system.

DOS Session breakpoint. A mechanism to regain control from a DOS Session; a V86 IRET frame is edited to point to an illegal V86 instruction, and the original CS:IP is saved in a table indexed by the address of the illegal instruction. A DOS Session breakpoint is a byte of memory visible in each DOS Session containing an illegal instruction.

double-byte character set (DBCS). A set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese and Korean which contain more characters than can be represented by 256 code points require double-byte character sets. As each character requires two bytes, the entering, displaying and printing of DBCS characters requires hardware and software that can support DBCS.

dragging. In computer graphics, moving an object on the display screen as if it were attached to the pointer.

drop. To fix the position of an object that is being dragged by releasing the select button of the pointing device.

DTL. See *dialog tag language*.

E

EBCDIC. Extended binary-coded decimal interchange code.

EGA. Extended graphics adapter.

entry field. A panel element in which users type information. Compare to *selection field*.

entry panel. A defined panel type containing one or more entry fields and protected information such as headings, prompts and explanatory text.

Event semaphore. A signaling mechanism, which enables a thread or process to notify waiting threads or processes that an event has occurred.

extended help. A facility that provides users with information about an entire application panel rather than a particular item on the panel.

entry-field control. The means by which the application receives data entered by the user in an entry field. When it has the input focus, it displays a flashing pointer at the position where the next typed character will go.

exit. The action that terminates the current function and returns the user to a higher level function. Repeated exit requests return the user to the point from which all functions provided to the system are accessible. Contrast with *cancel*.

extended-choice selection. A mode that allows the user to select more than one item from a window. Not all windows allow extended choice selection. Contrast with *multiple-choice selection*.

F

field-level help. Information specific to the field on which the cursor is positioned. This help function is "contextual" because it provides information about a specific item as it is currently used. The information is dependent upon the context within the work session.

File Manager. In the Presentation Manager, a program that displays directories and files and allows various actions on them.

file specification. The full identifier for a file which includes its file name, extension, path and drive.

font. A particular size and style of typeface that contains definitions of character sets, marker sets and pattern sets.

foreground program. The program with which the user is currently interacting. Also known as *interactive program*.

frame. The part of a window that can contain several different visual elements specified by the application but drawn and controlled by the Presentation Manager. The frame encloses the client area.

frame styles. Different standard window layouts provided by the Presentation Manager.

full-screen application. An application program that occupies the whole screen.

function key. A key that causes a specified sequence of operations to be performed when it is pressed; for example, F1 and Alt-K.

G

Global Descriptor Table (GDT). Defines code and data segments available to all tasks in an application.

glyph. A graphic symbol whose appearance conveys information.

GPI. Graphics Programming Interface. The formally defined programming language that is between an IBM graphics program and the user of the program. See also *API*.

graphics. A picture defined in terms of graphic primitives and graphics attributes.

graying. The indication that a choice on a pull-down is unavailable.

group. A collection of logically-connected controls. For example, the buttons controlling paper size for a printer. See also *program group*.

H

handle. An identifier that represents an object, such as a device or window.

hardcopy. Physical output (such as paper, slides or transparencies) from a device.

hard error. An error condition that requires either that the system be reconfigured or that the source of the error be removed before the system can resume reliable operation.

heap. An area of free storage available for dynamic allocation by the program. Its size varies depending on the storage requirements of the program.

help. A function that provides information about a specific field, an application panel or information about the help facility. It provides field help when the cursor is on a selection or entry field in an application panel or another help panel. It provides information about the application panel, called *extended help,* when the cursor is not in an interactive field.

help index. A facility that allows the user to select topics for which help is available.

help panel. A panel with information to assist users that is displayed in response to a help request from the user.

help window. A Common User Access defined secondary window that displays information when the user requests help.

hit testing. The means of identifying which window is associated with which input device event.

hook. A mechanism by which procedures are called when certain events occur in the system. For example, the filtering of mouse and keyboard input before it is received by an application program.

hook chain. A sequence of hook procedures that are "chained" together so that each event is passed in turn to each procedure in the chain.

hotspot. The part of the pointer that must touch an object before it can be selected. This is usually the tip of the pointer. Contrast with *action point*.

I

icon. A pictorial representation of an item the user can select. Icons can represent items (such as a document file) that the user wants to work on and actions that the user wants to perform. In the Presentation Manager, icons are used for data objects, system actions and minimized programs.

Icon Editor. The Presentation Manager-provided tool for creating icons.

information device context. A logical description of a data destination other than the screen (for example, a printer or plotter) but where no output will occur. Its purpose is to satisfy queries. See also *device context*.

input focus. The area of the screen that will receive input from an input device (typically the keyboard).

input router. OS/2 internal process that removes messages from the system queue.

interactive graphics. Graphics that can be moved or manipulated by a user at a terminal.

interactive program. A program that is running (active) and is ready to receive (or is receiving) input from the user. Also known as a *foreground program*. Compare with *active program* and contrast with *noninteractive program*.

Interrupt Request Flag. A bit in the 8259 PIC controller that indicates an interrupt is pending on particular level. The VPIC also maintains a virtual *interrupt request* flag for each interrupt level for each DOS Session.

Interrupt Service Flag. A bit in the 8259 PIC controller that indicates an interrupt request is being serviced. It is cleared when the PIC is sent EOI. The VPIC maintains a virtual *interrupt service* flag indicating that a simulated interrupt is in-progress in a DOS Session.

IOPL. Input/Output Privilege Level. Allows part of a Ring 3 application or device driver to execute at Ring 0.

IOPM. Input/Output Permission Map. Bitmap pointed to by the 386 TSS that controls, on a per-port basis, whether an IN/OUT instruction causes a fault.

IRQ. A term that broadly means an *interrupt request level* This term refers to either pending or in-service interrupt requests, or to a specific level (for example, IRQ 4).

J

journal. A special-purpose file that is used to record changes made in the system.

K

kanji. A graphic character set used in Japanese ideographic alphabets.

kerning. The design of graphics characters so that their character boxes overlap. Used to space text proportionally.

keys help. A facility that gives users a listing of all the key assignments for the current application.

L

language support procedure. Function provided by the Presentation Interface for applications that do not or cannot (as in the case of COBOL and FORTRAN programs) provide their own dialog or window procedures.

LIFO Stack. A data structure from which data is retrieved in last-in, first-out order.

linked list. A list in which the data elements may be dispersed but in which each data element contains information for locating the next. Synonym for *chained list*.

list box. A control window containing a vertical list of selectable descriptions.

list panel. A defined panel type that displays a list of items from which users can select one or more choices and then specify one or more actions to work on those choices.

Local Descriptor Table (LDT). Defines code and data segments specific to a single task.

LVB. Logical Video Buffer.

M

main window. The window that is positioned relative to the desktop window.

maximize. A window-sizing action that makes the window the largest size possible.

media window. The part of the physical device (display, printer or plotter) on which a picture is presented.

memory device context. A logical description of a data destination that is a memory bitmap. See also *device context*.

menu. A type of panel that consists of one or more selection fields. Also called a *menu panel*.

message. 1. In Presentation Manager, a packet of data used for communication between the Presentation Interface and windowed applications.

2. In a user interface, information not requested by users but presented to users by the computer in response to a user action or internal process.

Messages on status, problems or user actions from a computer application should be distinguished from a "message" or note sent to users by other users over a communications link.

message filter. The means of selecting which messages from a specific window will be handled by the application.

message queue. A sequenced collection of messages to be read by the application.

metafile. The generic name for the definition of the contents of a picture. Metafiles are used to allow pictures to be used by other applications.

metafile device context. A logical description of a data destination that is a metafile which is used for graphics interchange. See also *device context*.

metalanguage. A language used to specify another language.

mickey. A unit of measurement for physical mouse motion whose value depends on the mouse device driver currently loaded.

minimize. A window-sizing action that makes the window the smallest size possible. In the Presentation Manager, minimized windows are represented by icons.

mixed character string. A string containing a mixture of one-byte and *kanji* or *hangeul* (two-byte) characters.

mnemonic. A method of selecting an item on a pull-down by means of typing the highlighted letter in the menu item.

modal dialog box. The type of control that allows the operator to perform input operations on only the current dialog box or one of its child windows. Also known as a *serial dialog box*. Contrast with *parallel dialog box*.

modeless dialog box. The type of control that allows the operator to perform input operations on any of the application's windows. Also known as a *parallel dialog box*. Contrast with *modal dialog box*.

mouse. A hand-held device that is moved around to position the pointer on the screen.

Multiple DOS Session. A system service that coordinates the concurrent operation of multiple DOS Sessions. See also *DOS Session*.

Multiple Virtual DOS Machine. Deprecated term for multiple DOS Sessions.

multiple-choice selection. A mode that allows users to select any number of choices including none at all. See also *check box*. Contrast with *extended-choice selection*.

multi-tasking. The concurrent processing of applications or parts of applications. A running application and its data are protected from other concurrently running applications.

N

named pipe. A named object that provides client-to-server, server-to-client or duplex communication between unrelated processes. Contrast with *unnamed pipe*.

noninteractive program. A program that is running (active) but is not ready to receive input from the user. Compare with *active program* and contrast with *interactive program*.

null-terminated string. A string of (n + 1) characters where the (n + 1)th character is the 'null' character (X'00') and is used to represent an n-character string with implicit length. Also known as 'zero-terminated' string and 'ASCIIZ' string.

O

object window. A window that does not have a parent but which may have child windows. An object window cannot be presented on a device.

open. To start working with a file, directory or other object.

output area. The area of the output device within which the picture is to be displayed, printed or plotted.

owner window. A window into which specific events that occur in another (owned) window are reported.

P

paint. The action of drawing or redrawing the contents of a window.

panel. A particular arrangement of information grouped together for presentation to the user in a window.

panel area. An area within a panel that contains related information. The three major panel areas defined by the Common User Access are the action bar, the function key area and the panel body.

panel body. The portion of a panel not occupied by the action bar, function key area, title or scroll bars. The panel body may contain protected information, selection fields and entry fields. The layout and content of the panel body determine the panel type.

panel body area. The part of a window not occupied by the action bar or function key area. The panel body area may contain information, selection fields and entry fields. Also known as *client area*.

panel body area separator. A line or color boundary that provides users with a visual distinction between two adjacent areas of a panel.

panel definition. A description of the contents and characteristics of a panel. Thus, a panel definition is the application developer's mechanism for predefining the format to be presented to users in a window.

panel ID. A panel element located in the upper left-hand corner of a panel body that identifies that particular panel within the application.

panel title. A panel element that identifies the information in the panel.

paper size. The size of paper, defined in either standard U.S. or European names (for example, A, B, A4) and measured in inches or millimeters respectively.

parallel dialog box. See *modeless dialog box*.

parent window. The window relative to which one or more child windows are positioned. Contrast with *child window*.

pel. The smallest area of a display screen capable of being addressed and switched between visible and invisible states. Synonymous with *display point*, *pixel* and *picture element*.

physical device driver (PDD). Device driver responsible for primary control of a physical device. This corresponds very closely to the OS/2 1.00 dual-mode device driver, except that it does not support the DOS environment directly. Rather, it supports interfaces that can be used by virtual device drivers to support &dosapps..

pick. To select part of a displayed object using the pointer.

pipe. See *named pipe*, *unnamed pipe*.

pixel. Synonym for *pel*. See *pel*

plotter. An output device that uses pens to draw its output on paper or transparency foils.

pointer. The symbol displayed on the screen that is moved by a pointing device such as a *mouse*. The pointer is used to point at items that users can select. Contrast with *cursor*.

pointing device. A device (such as a mouse) used to move a pointer on the screen.

pointings. Pairs of x-y coordinates produced by an operator defining positions on a screen with a pointing device such as a *mouse*.

polyfillet. A curve based on a sequence of lines. It is tangential to the end points of the first and last lines and tangential also to the midpoints of all other lines. See also *fillet*.

polyline. A sequence of adjoining lines.

pop. To retrieve an item from a last-in-first-out stack of items. Contrast with *push*.

pop-up window. A window that appears on top of another window in a dialog. Each pop-up window must be completed before returning to the underlying window.

Presentation Manager. The OS/2 Control Program plus the visual component that presents, in windows, a graphics-based interface to applications and files installed and running in OS/2.

primary window. The window in which the main dialog between users and the application takes place. In a multi-programming environment, each application starts in its own primary window. The primary window remains for the duration of the application although the panel displayed will change as the user's dialog moves forward. See also *secondary window*.

print job. The result of sending a document or picture to be printed.

Print Manager. In the Presentation Manager, the part of the spooler that manages the spooling process. It also allows users to view print queues and to manipulate print jobs.

process. An instance of an executing application and the resources it is using.

program details. Information about a program that is specified in the Desktop Manager window and is used when the program is started.

program group. In the Presentation Manager, several programs that can be acted upon as a single entity.

program name. The full file specification of a program. Contrast with *program title*.

program title. The name of a program as it is listed in the Desktop Manager window. Contrast with *program name*.

pull-down. An extension of the *action bar* that displays a list of choices available for a selected action bar choice. After users select an action bar choice, the pull-down appears with the list of choices. Additional *pop-up windows* may appear from pull-down choices to further extend the actions available to users.

push. To add an item to a last-in-first-out stack of items. Contrast with *pop*.

pushbutton. A control window shaped like a rounded-corner rectangle and containing text, that invokes an immediate action such as 'enter' or 'cancel'.

Q

queue. A list of print jobs waiting to be printed.

queued device context. A logical description of a data destination (for example, a printer or plotter) where the output is to go through the spooler. See also *device context*.

R

radio button. A control window shaped like a round button on the screen that can be in a checked or unchecked state. It is used to select a single item from list. Contrast with *check box*.

reentrant. The attribute of a program or routine that allows the same copy of the program or routine to be used concurrently by two or more tasks.

reference phrase. A word or phrase that is emphasized in a device-dependent manner in order to inform the user that additional information for the word or phrase is available.

reference phrase help. Provides help information on a selectable phrase.

refresh. To update a window with changed information to its current status.

resource. The means of providing extra information used in the definition of a window. A resource can contain definitions of fonts, templates, accelerators and mnemonics; the definitions are held in a resource file.

restore. To return a window to its original size or position following a sizing or moving action.

reverse video. A form of alphanumeric highlighting for a character, field or cursor in which its color is exchanged with that of its background. For example, changing a red character on a black background to a black character on a red background.

RGB. Red-green-blue. For example "RGB display".

roman. Relating to a type style with upright characters.

S

screen. The physical surface of a workstation or terminal upon which information is presented to users.

screen device context. A logical description of a data destination that is a particular window on the screen. See also *device context*.

scroll bar. A control window, horizontally or vertically aligned, that allows the user to scroll additional data into an associated panel area.

scrollable entry field. An entry field larger than the visible field.

scrollable selection field. A selection field that contains more choices than are visible.

scrolling. Moving a display image vertically or horizontally in a manner such that new data appears at one edge as existing data disappears at the opposite edge.

secondary window. A type of window associated with the primary window in a dialog. A secondary window begins a secondary and parallel dialog that runs at the same time as the primary dialog.

segment. See *graphics segment*.

select. To mark or choose an item. Note that *select* means to mark or type in a choice on the screen; *enter* means to send all selected choices to the computer for processing.

select button. The button on a pointing device, such as a mouse, that is pressed to select a menu choice. Also known as button 1.

selection cursor. A type of cursor used to indicate the choice or entry field users want to interact with. It is represented by highlighting the item it is currently positioned on.

selection field. A field containing a list of choices from which the user can select one or more.

semaphore. An object used by multi-threaded applications for signaling purposes and for controlling access to serially reusable resources.

separator. See *panel body area separator*.

serial dialog box. See *modal dialog box*.

serially reusable resource (SRR). A logical resource or object that can be accessed by only one task at a time.

session. A routing mechanism for user interaction via the console.

shadow box. The area on the screen that follows mouse movements and shows what shape the window will take if the mouse button is released.

shutdown. In the Desktop Manager, the procedure required before the computer is switched off to ensure that data is not lost.

sibling windows. Child windows that have the same parent window.

slider box. An area on the scroll bar that indicates the size and position of the visible information in a panel area in relation to the information available. Also known as *thumb mark*.

spline. A sequence of one or more Bezier curves.

spooler. A program that intercepts the data going to printer devices and writes it to disk. The data is printed or plotted when it is complete and the required device

is available. The spooler prevents output from different sources being intermixed.

standard window. A collection of windows that form a panel.

static control. The means by which the application presents descriptive information (for example, headings and descriptors) to the user. The user cannot change this information.

suballocation. The allocation of a part of one extent for occupancy by elements of a component other than the one occupying the remainder of the extent.

switch. An action that moves the input focus from one area to another. This can be within the same window or from one window to another.

switch list. See Task List.

symbolic identifier. A text string that equates to an integer value in an include file that is used to identify a programming object.

System Menu. In the Presentation Manager, the pull-down in the top left corner of a window that allows it to be moved and sized with the keyboard.

system queue. This is the master queue for all pointer device or keyboard events.

Systems Application Architecture. A formal set of rules that enables applications to be run without modification in different computer environments.

T

tag. A markup language word and its attributes that are entered in the source file to identify parts of a panel or other dialog objects.

Task List. In the Presentation Manager, the list of programs that are active. The list can be used to switch to a program and to stop programs.

template. An ASCII-text definition of an action bar and pull-down menu held in a resource file or as a data structure in program memory.

text. Characters or symbols.

text cursor. A symbol displayed in an entry field that indicates where typed input will appear.

text window. Also known as the VIO window. The environment in which OS/2 runs AVIO applications.

thread. A unit of execution within a process.

thumb mark. The portion of the scroll bar that describes the range and properties of the data that is currently visible in a window. Also known as a *slider box*.

tilde. A mark used to denote the character that is to be used as a mnemonic when selecting text items within a menu.

title bar. The area at the top of a window that contains the window title. The title bar is highlighted when that window has the input focus. Contrast with *panel title*.

Tree. In the Presentation Manager, the window in the File Manager that shows the organization of drives and directories.

U

unnamed pipe. A circular buffer created in memory; used by related processes to communicate with one another. Contrast with *named pipe*.

User Shell. A component of OS/2 that uses a graphics-based, windowed interface to allow the user to manage applications and files installed and running under OS/2.

V

VGA. Video graphics array.

VIO. Video Input/Output.

virtual device driver (VDD). Maintains shadow state of hardware, if necessary. Allows a DOS Session to execute *in a window* or *in the background* by intercepting direct device access and simulating that device.

Virtual DevHlp (VDH). Kernel (linear memory, paging, hardware interrupt, event control, port control) services provided to virtual device drivers.

Virtual DOS Machine. Depreciated term for DOS Session. See *DOS Session*.

virtual memory (VM). Addressable space that is apparent to the user as the processor storage space but not having a fixed physical location.

Virtual Programmable Interrupt Controller (VPIC). Virtualizes the 8259 Programmable Interrupt Controller (PIC). A special virtual device driver, in that it provides services to other virtual device drivers.

---

* Trademark of the IBM Corporation

visible region. A window's presentation space clipped to the boundary of the window and the boundaries of any overlying window.

V86 Mode. Virtual 8086 mode of the 80386.

W

wildcard character. The global file-name characters ? or *.

window. A rectangular area of the screen through which a panel or portion of a panel is displayed. A window can be smaller than or equal in size to the screen. Windows can overlap on the screen and give the appearance of one window being on top of another.

window class. The grouping of windows whose processing needs conform to the services provided by one window procedure.

window coordinates. The means by which a window position or size is defined; measured in device units or *pels*.

window procedure. Code that is activated in response to a message.

window rectangle. The means by which the size and position of a window is described in relation to the desktop window.

window style. The set of properties that influence how events related to a particular window will be processed.

workstation. A display screen together with attachments such as a keyboard, a local copy device or a tablet.

WYSIWYG. What You See Is What You Get. A capability that enables text to be displayed on a screen in the same way that it will be formatted on a printer.

Z

z-order. The order in which sibling windows are presented. The topmost sibling window obscures any portion of the siblings that it overlaps; the same effect occurs down through the order of lower sibling windows.

zooming. In graphics applications, the process of increasing or decreasing the size of picture.

Index

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Numerics

International Business
Machines Corporation

Printed in the
United States of America

10G6310

S10G-6310-00

P10G6310